WIKIPEDIA

Virtual machine

In computing, a **virtual machine** (**VM**) is the virtualization or emulation of a computer system. Virtual machines are based on computer architectures and provide the functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination of the two. Virtual machines differ and are organized by their function, shown here:

- **System virtual machines** (also called full virtualization VMs) provide a substitute for a real machine. They provide the functionality needed to execute entire operating systems. A hypervisor uses native execution to share and manage hardware, allowing for multiple environments that are isolated from one another yet exist on the same physical machine. Modern hypervisors use hardware-assisted virtualization, with virtualization-specific hardware features on the host CPUs providing assistance to hypervisors.
- **Process virtual machines** are designed to execute computer programs in a platform-independent environment.

Some virtual machine emulators, such as QEMU and video game console emulators, are designed to also emulate (or "virtually imitate") different system architectures, thus allowing execution of software applications and operating systems written for another CPU or architecture. OS-level virtualization allows the resources of a computer to be partitioned via the kernel. The terms are not universally interchangeable.

Definitions

System virtual machines

A "virtual machine" was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real computer machine."[1] Current use includes virtual machines that have no direct correspondence to any real hardware.[2] The physical, "real-world" hardware running the VM is generally referred to as the 'host', and the virtual machine emulated on that machine is generally referred to as the 'guest'. A host can emulate several guests, each of which can emulate different operating systems and hardware platforms.

The desire to run multiple operating systems was the initial motive for virtual machines, so as to allow time-sharing among several single-tasking operating systems. In some respects, a system virtual machine can be considered a generalization of the concept of virtual memory that historically preceded it. IBM's CP/CMS, the first systems to allow full virtualization, implemented time sharing by providing each user with a single-user operating system, the Conversational Monitor System (CMS). Unlike virtual memory, a system virtual machine entitled the user to write privileged instructions in their code. This approach had certain advantages, such as adding input/output devices not allowed by the standard system.[2]

As technology evolves virtual memory for purposes of virtualization, new systems of memory overcommitment may be applied to manage memory sharing among multiple virtual machines on one computer operating system. It may be possible to share *memory pages* that have identical contents among multiple virtual machines that run on the same physical machine, what may result in mapping them to the same physical page by a technique termed kernel same-page merging (KSM). This is especially useful for read-only pages, such as those holding code segments, which is the case for multiple virtual machines running the same or similar software, software libraries, web servers, middleware components, etc. The guest operating systems do not need to be compliant with the host hardware, thus making it possible to run different operating systems on the same computer (e.g., Windows, Linux, or prior versions of an operating system) to support future software.[3]

The use of virtual machines to support separate guest operating systems is popular in regard to embedded systems. A typical use would be to run a real-time operating system simultaneously with a preferred complex operating system, such as Linux or Windows. Another use would be for novel and unproven software still in the developmental stage, so it runs inside a sandbox. Virtual machines have other advantages for operating system development and may include improved debugging access and faster reboots.[4]

Multiple VMs running their own guest operating system are frequently engaged for server consolidation.[5]

Process virtual machines

A process VM, sometimes called an *application virtual machine*, or *Managed Runtime Environment* (MRE), runs as a normal application inside a host OS and supports a single process. It is created when that process is started and destroyed when it exits. 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.

A process VM provides a high-level abstraction – that of a high-level programming language (compared to the low-level ISA abstraction of the system VM). Process VMs are implemented using an interpreter; performance comparable to compiled programming languages can be achieved by the use of just-in-time compilation.

This type of VM has become popular with the Java programming language, which is implemented using the Java virtual machine. Other examples include the Parrot virtual machine and the .NET Framework, which runs on a VM called the Common Language Runtime. All of them can serve as an abstraction layer for any computer language.

A special case of process VMs are systems that abstract over the communication mechanisms of a (potentially heterogeneous) computer cluster. Such a VM does not consist of a single process, but one process per physical machine in the cluster. They are designed to ease the task of programming concurrent applications by letting the programmer focus on algorithms rather than the communication mechanisms provided by the interconnect and the OS. They do not hide the fact that communication takes place, and as such do not attempt to present the cluster as a single machine.

Unlike other process VMs, these systems do not provide a specific programming language, but are embedded in an existing language; typically such a system provides bindings for several languages (e.g., C and Fortran). Examples are Parallel Virtual Machine (PVM) and Message Passing Interface (MPI).

History

Both system virtual machines and process virtual machines date to the 1960s and remain areas of active development.

System virtual machines grew out of time-sharing, as notably implemented in the Compatible Time-Sharing System (CTSS). Time-sharing allowed multiple users to use a computer concurrently: each program appeared to have full access to the machine, but only one program was executed at the time, with the system switching between programs in time slices, saving and restoring state each time. This evolved into virtual machines, notably via IBM's research systems: the M44/44X, which used partial virtualization, and the CP-40 and SIMMON, which used full virtualization, and were early examples of hypervisors. The first widely available virtual machine architecture was the CP-67/CMS (see History of CP/CMS for details). An important distinction was between using multiple virtual machines on one host system for time-sharing, as in M44/44X and CP-40, and using one virtual machine on a host system for prototyping, as in SIMMON. Emulators, with hardware emulation of earlier systems for compatibility, date back to the IBM System/360 in 1963,[6][7] while the software emulation (then-called "simulation") predates it.

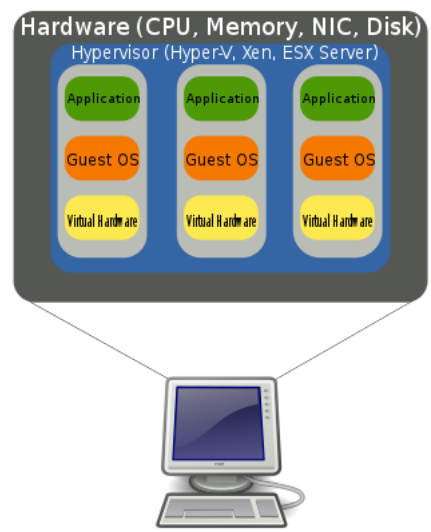
Process virtual machines arose originally as abstract platforms for an intermediate language used as the intermediate representation of a program by a compiler; early examples date to around 1966. An early 1966 example was the O-code machine, a virtual machine that executes O-code (object code) emitted by the front end of the BCPL compiler. This abstraction allowed the compiler to be easily ported to a new architecture by implementing a new back end that took the existing O-code and compiled it to machine code for the underlying physical machine. The Euler language used a similar design, with the intermediate language named *P* (portable).[8] This was popularized around 1970 by Pascal, notably in the Pascal-P system (1973) and Pascal-S compiler (1975), in which it was termed p-code and the resulting machine as a p-code machine. This has been influential, and virtual machines in this sense have been often generally

called p-code machines. In addition to being an intermediate language, Pascal p-code was also executed directly by an interpreter implementing the virtual machine, notably in UCSD Pascal (1978); this influenced later interpreters, notably the Java virtual machine (JVM). Another early example was SNOBOL4 (1967), which was written in the SNOBOL Implementation Language (SIL), an assembly language for a virtual machine, which was then targeted to physical machines by transpiling to their native assembler via a macro assembler.[9] Macros have since fallen out of favor, however, so this approach has been less influential. Process virtual machines were a popular approach to implementing early microcomputer software, including Tiny BASIC and adventure games, from one-off implementations such as Pyramid 2000 to a general-purpose engine like Infocom's z-machine, which Graham Nelson argues is "possibly the most portable virtual machine ever created".[10]

Significant advances occurred in the implementation of Smalltalk-80,[11] particularly the Deutsch/Schiffmann implementation[12] which pushed just-in-time (JIT) compilation forward as an implementation approach that uses process virtual machine.[13] Later notable Smalltalk VMs were VisualWorks, the Squeak Virtual Machine,[14] and Strongtalk.[15] A related language that produced a lot of virtual machine innovation was the Self programming language,[16] which pioneered adaptive optimization[17] and generational garbage collection. These techniques proved commercially successful in 1999 in the HotSpot Java virtual machine.[18] Other innovations include a register-based virtual machine, to better match the underlying hardware, rather than a stack-based virtual machine, which is a closer match for the programming language; in 1995, this was pioneered by the Dis virtual machine for the Limbo language.

Full virtualization

In full virtualization, the virtual machine simulates enough hardware to allow an unmodified "guest" OS (one designed for the same instruction set) to be run in isolation. This approach was pioneered in 1966 with the IBM CP-40 and CP-67, predecessors of the VM family.



Logical diagram of full virtualization

Examples outside the mainframe field include Parallels Workstation, Parallels Desktop for Mac, VirtualBox, Virtual Iron, Oracle VM, Virtual PC, Virtual Server, Hyper-V, VMware Fusion, VMware Workstation, VMware Server (discontinued, formerly called GSX Server), VMware ESXi, QEMU, Adeos, Mac-on-Linux, Win4BSD, Win4Lin Pro, and Egenera vBlade technology.

Hardware-assisted virtualization

In hardware-assisted virtualization, the hardware provides architectural support that facilitates building a virtual machine monitor and allows guest OSes to be run in isolation.[19] Hardware-assisted virtualization was first introduced on the IBM System/370 in 1972, for use with VM/370, the first virtual machine operating system offered by IBM as an official product.[20]

In 2005 and 2006, Intel and AMD provided additional hardware to support virtualization. Sun Microsystems (now Oracle Corporation) added similar features in their UltraSPARC T-Series processors in 2005. Examples of virtualization platforms adapted to such hardware include KVM, VMware Workstation, VMware Fusion, Hyper-V, Windows Virtual PC, Xen, Parallels Desktop for Mac, Oracle VM Server for SPARC, VirtualBox and Parallels Workstation.

In 2006, first-generation 32- and 64-bit x86 hardware support was found to rarely offer performance advantages over software virtualization.[21]

OS-level virtualization

In OS-level virtualization, a physical server is virtualized at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server. The "guest" operating system environments share the same running instance of the operating system as the host system. Thus, the same operating system kernel is also used to

implement the "guest" environments, and applications running in a given "guest" environment view it as a stand-alone system. The pioneer implementation was FreeBSD jails; other examples include Docker, Solaris Containers, OpenVZ, Linux-VServer, LXC, AIX Workload Partitions, Parallels Virtuozzo Containers, and iCore Virtual Accounts.

See also

- Amazon Machine Image
- Desktop virtualization
- Linux containers
- Native development kit
- Paravirtualization
- Storage hypervisor
- Universal Turing machine
- Virtual appliance
- Virtual backup appliance
- Virtual disk image
- Virtual DOS machine (VDM)
- Virtual machine escape

References

1. Popek, Gerald J.; Goldberg, Robert P. (1974). "Formal requirements for virtualizable third generation architectures" (https://www.cse.iitb.ac.in/~puru/courses/spring12/cs695/downloads/popekgoldberg.pdf) (PDF). *Communications of the ACM*. **17** (7): 412–421. doi:10.1145/361011.361073 (https://doi.org/10.1145%2F361011.361073). S2CID 12680060 (https://api.semanticscholar.org/CorpusID:12680060).
2. Smith, James E.; Nair, Ravi (2005). "The Architecture of Virtual Machines" (http://digital.library.wisc.edu/1793/11154). *Computer*. **38** (5): 32–38, 395–396. doi:10.1109/MC.2005.173 (https://doi.org/10.1109%2FMC.2005.173). S2CID 6578280 (https://api.semanticscholar.org/CorpusID:6578280).
3. Oliphant, Patrick. "Virtual Machines" (https://web.archive.org/web/20160729182221/http://www.virtualcomputing.net/virtual-machines). VirtualComputing. Archived from the original (http://www.virtualcomputing.net/virtual-machines) on 2016-07-29. Retrieved 2015-09-23. "Some people use that capability to set up a separate virtual machine running Windows on a Mac, giving them access to the full range of applications available for both platforms."
4. "Super Fast Server Reboots – Another reason Virtualization rocks" (https://web.archive.org/web/20060614120104/http://www.vmwarez.com/2006/05/super-fast-server-reboots-another.html). *vmwarez.com*. 2006-05-09. Archived from the original (http://www.vmwarez.com/2006/05/super-fast-server-reboots-another.html) on 2006-06-14. Retrieved 2013-06-14.
5. "Server Consolidation and Containment With Virtual Infrastructure" (http://www.vmware.com/pdf/server_consolidation.pdf) (PDF). VMware. 2007. Archived (https://web.archive.org/web/20131228142508/http://www.vmware.com/pdf/server_consolidation.pdf) (PDF) from the original on 2013-12-28. Retrieved 2015-09-29.
6. Pugh, Emerson W. (1995). *Building IBM: Shaping an Industry and Its Technology* (https://archive.org/details/buildingibmshapi00pugh). MIT. p. 274 (https://archive.org/details/buildingibmshapi00pugh/page/n280). ISBN 978-0-262-16147-3.
7. Pugh, Emerson W.; et al. (1991). *IBM's 360 and Early 370 Systems* (https://archive.org/details/ibms360early370s0000pugh). MIT. pp. 160–161 (https://archive.org/details/ibms360early370s0000pugh/page/160). ISBN 978-0-262-16123-7.
8. Wirth, Niklaus Emil; Weber, Helmut (1966). *EULER: a generalization of ALGOL, and its formal definition: Part II, Communications of the Association for Computing Machinery* (http://dl.acm.org/citation.cfm?doid=365170.365202,). Vol. 9. New York: ACM. pp. 89–99.
9. Griswold, Ralph E. *The Macro Implementation of SNOBOL4*. San Francisco, CA: W. H. Freeman and Company, 1972 (ISBN 0-7167-0447-1), Chapter 1.
10. Nelson, Graham A. "About Interpreters" (http://www.inform-fiction.org/zmachine/interpreters.html). *Inform website*. Archived (https://web.archive.org/web/20091203031858/http://www.inform-fiction.org/zmachine/interpreters.html) from the original on 2009-12-03. Retrieved 2009-11-07.
11. Goldberg, Adele; Robson, David (1983). *Smalltalk-80: The Language and its Implementation* (https://archive.org/details/smalltalk80langu00gold). Addison-Wesley Series in Computer Science. Addison-Wesley. ISBN 978-0-201-11371-6.
12. Deutsch, L. Peter; Schiffman, Allan M. (1984). "Efficient implementation of the Smalltalk-80 system" (http://portal.acm.org/citation.cfm?id=800017.800542). *POPL*. Salt Lake City, Utah: ACM. doi:10.1145/800017.800542 (https://doi.org/10.1145%2F800017.800542). ISBN 0-89791-125-3.

13. Aycock, John (2003). "A brief history of just-in-time". *ACM Comput. Surv.* **35** (2): 97–113. doi:10.1145/857076.857077 (https://doi.org/10.1145%2F857076.857077). S2CID 15345671 (https://api.semanticscholar.org/CorpusID:15345671).
14. Ingalls Jr., Daniel "Dan" Henry Holmes; Kaehler, Ted; Maloney, John; Wallace, Scott; Kay, Alan Curtis (1997). "Back to the future: the story of Squeak, a practical Smalltalk written in itself". *OOPSLA '97: Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, US: ACM Press. pp. 318–326. doi:10.1145/263698.263754 (https://doi.org/10.1145%2F263698.263754). ISBN 0-89791-908-4.
15. Bracha, Gilad; Griswold, David (1993). "Strongtalk: Typechecking Smalltalk in a Production Environment". *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*. OOPSLA '93. New York, NY, US: ACM. pp. 215–230. doi:10.1145/165854.165893 (https://doi.org/10.1145%2F165854.165893). ISBN 978-0-89791-587-8.
16. Ungar, David Michael; Smith, Randall B. (December 1987). "Self: The power of simplicity". *ACM SIGPLAN Notices*. **22** (12): 227–242. doi:10.1145/38807.38828 (https://doi.org/10.1145%2F38807.38828). ISSN 0362-1340 (https://www.worldcat.org/issn/0362-1340).
17. Hölzle, Urs; Ungar, David Michael (1994). "Optimizing dynamically-dispatched calls with run-time type feedback" (http://portal.acm.org/citation.cfm?id=178243.178478). *PLDI*. Orlando, Florida, United States: ACM. pp. 326–336. doi:10.1145/178243.178478 (https://doi.org/10.1145%2F178243.178478). ISBN 0-89791-662-X.
18. Paleczny, Michael; Vick, Christopher; Click, Cliff (2001). "The Java HotSpot server compiler" (http://portal.acm.org/citation.cfm?id=1267848). *Proceedings of the Java Virtual Machine Research and Technology Symposium on Java Virtual Machine Research and Technology Symposium*. Vol. 1. Monterey, California: USENIX Association.
19. Uhlig, Rich; Neiger, Gil; Rodgers, Dion; Santoni, Amy L.; Martins, Fernando C. M.; Anderson, Andrew V.; Bennett, Steven M.; Kägi, Alain; Leung, Felix H.; Smith, Larry (May 2005). "Intel virtualization technology". *Computer*. **38** (5): 48–56. doi:10.1109/MC.2005.163 (https://doi.org/10.1109%2FMC.2005.163). S2CID 18514555 (https://api.semanticscholar.org/CorpusID:18514555).
20. Randal, A. (2019). The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers.
21. Adams, Keith; Agesen, Ole (2006-10-21). *A Comparison of Software and Hardware Techniques for x86 Virtualization* (http://www.vmware.com/pdf/asplos235_adams.pdf) (PDF). ASPLOS'06 21–25 October 2006. San Jose, California, US. Archived (https://web.archive.org/web/20100820201944/http://www.vmware.com/pdf/asplos235_adams.pdf) (PDF) from the original on 2010-08-20. "Surprisingly, we find that the first-generation hardware support rarely offers performance advantages over existing software techniques. We ascribe this situation to high VMM/guest transition costs and a rigid programming model that leaves little room for software flexibility in managing either the frequency or cost of these transitions."

Further reading

- James E. Smith, Ravi Nair, *Virtual Machines: Versatile Platforms For Systems And Processes*, Morgan Kaufmann, May 2005, ISBN 1-55860-910-5, 656 pages (covers both process and system virtual machines)
- Craig, Iain D. *Virtual Machines*. Springer, 2006, ISBN 1-85233-969-1, 269 pages (covers only process virtual machines)

External links

- Mendel Rosenblum (2004-08-31). "The Reincarnation of Virtual Machines" (https://queue.acm.org/detail.cfm?id=1017000). *ACM Queue*. Vol. 2, no. 5.
- Sandia National Laboratories Runs 1 Million Linux Kernels as Virtual Machines (http://www.net-security.org/secworld.php?id=7837)
- The design of the Inferno virtual machine by Phil Winterbottom and Rob Pike (http://doc.cat-v.org/inferno/4th_edition/dis_VM_design)