Virtualisierung Backstage

Wie funktioniert Virtualisierung und was eint oder trennt die zahlreichen Angebote? Welches eignet sich für welchen Zweck? Ein Blick hinter die Kulissen. Jens-Christoph Brendel

Heute tummelt sich auf dem Markt für Virtualisierungslösungen eine Vielzahl an Techniken und Produkten und es herrscht ein ziemliches Begriffsgewirr. So unterscheidet man nach dem Gegenstand Server-, Speicher- und Netzwerk-Virtualisierung. Allerdings überschneiden sich diese Formen häufig. Nach dem Einsatzzweck wird zuweilen Desktop- von Server-Virtualisierung getrennt, aber auch diese Grenze ist unscharf. Ein weiteres Kriterium ist, ob die Virtualisierung eher durch Hardware oder Software bewerkstelligt wird. Je nachdem, ob sie an eine Plattform gebunden oder plattformunabhängig ist, sprechen manche auch von In-the-Box- und Out-of-the-Box-Virtualisierung.

Ordnung tut Not. Bester Ausgangspunkt dafür ist ein ganz normaler Rechner, der mit Virtualisierung nichts zu schaffen hat. Bei ihm kann man ganz grob drei Komponenten unterscheiden: Hardware, Betriebssystem und Anwendungen. Hard- und Software verbindet eine Schnittstelle, die als Instruction Set Architecture (ISA) bezeichnet wird (**Abbildung 1**).

Die ISA definiert die Eigenschaften eines Rechners, die für den Programmierer nutzbar sind, beispielsweise den Befehlssatz der

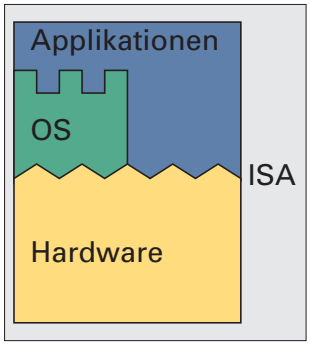

Abbildung 1: Die Instruction Set Architecture (ISA) bildet die Schnittstelle zwischen Soft- und Hardware und vereinbart, was ein Compiler voraussetzen darf.

CPU oder die verfügbaren Register des Prozessors. Man kann sie sich als eine Art Vertrag vorstellen, der dem Entwickler und seinem Compiler ein bestimmtes Verhalten der Hardware zusichert.

Virtualisierung fügt in dies Modell in jedem Fall eine Schicht oberhalb der ISA ein. Zuerst ist wesentlich, ob diese Virtualisierungsschicht sowohl Anwendung wie Betriebssystem des Gastes unterkellert, dann spricht man von System-Virtualisierung, oder nur unter der Anwendungsschicht liegt, dann ist von Prozess-Virtualisierung die Rede. Beide Klassen gliedern sich in einem zweiten Schritt danach, ob die Komponenten unter- und oberhalb der Virtualisierungsschicht dieselbe ISA verwenden oder nicht. In einem dritten Schritt schließlich ist nach der Lage dieser Schicht (**Abbildung 2**) zu unterscheiden.

Prozess-Virtualisierung

Mit Prozess-Virtualisierung – von der dann im Folgenden nicht weiter die Rede sein soll – hat man es eigentlich laufend zu tun. Besonders die Multiprogramming getaufte Unterart mit gleicher ISA ist so häufig, dass sie kaum noch auffällt. Denn dieser Technik bedient sich eigentlich jedes normale Betriebssystem, indem es den Prozessen vorspiegelt, sie könnten einen kompletten Rechner benutzen, während sie das Betriebssystem tatsächlich in ihrem Adressenraum gefangen hält und in der Regel nur über sein Ressourcenmanagement auf Hardware zugreifen lässt. Damit leben sie im Alltag immer in einem virtuellen Raum. Dabei ist implizit jedes Mal auch Speicher-Virtualisierung im Spiel, die in modernen Prozessoren die MMU erledigt.

Auch Prozess-Virtualisierung mit unterschiedlicher ISA kennt jeder aus eigener Erfahrung, etwa von Java und seiner virtuellen Maschine oder von Pascals P-Code. Das Java-Programm rechnet mit Funktion und Befehlssatz der JVM, die ihrerseits auf der ISA der Hardware aufbaut. Die Gattungsbezeichnung dafür lautet High Level Language Virtual Machine (HLL VM).

System-Virtualisierung

Interessanter wird es bei der Virtualisierung ganzer Systeme, die sich genau so differenzieren lässt. Im ersten und häufigsten Fall verwenden alle Betriebssysteme – der Host wie seine Gäste – dieselbe ISA. Der Virtualisierungslayer vervielfältigt diese Schnittstelle lediglich und fängt gleichzeitig die privilegierten Operationen der Gast-Betriebssysteme ab. Das ist nötig, weil die

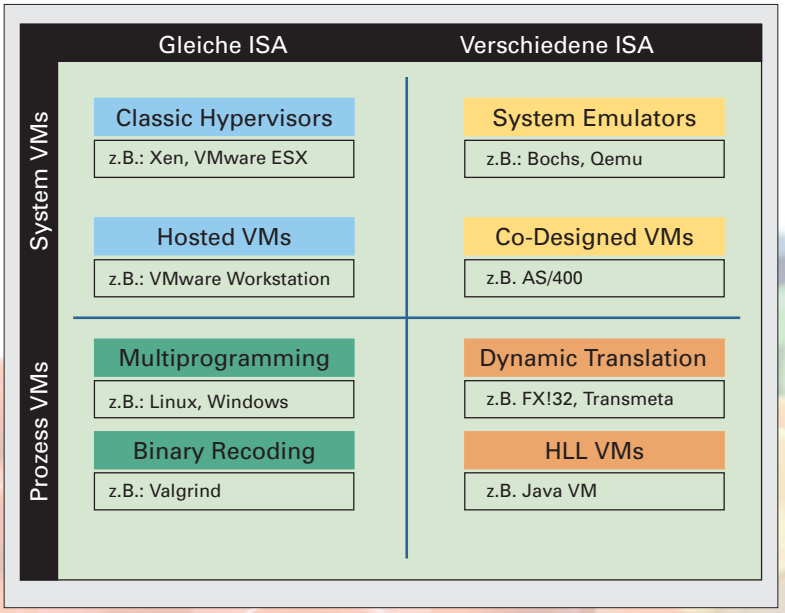

Abbildung 2: Die verschiedenen Arten virtueller Maschinen bilden vier große Klassen, unter denen die ersten drei der linken Seite die größte Anhängerschar gefunden haben.

Memory Management Unit (MMU): *Funktionseinheit moderner Prozessoren, die logische in physische Adressen übersetzt. Der logische Adressraum wird dabei in Gruppen aufeinander folgender Adressen (Seiten) aufgeteilt, der Umsetzungsprozess durch Caches (Translation Lookaside Buffer, TLB) beschleunigt.*

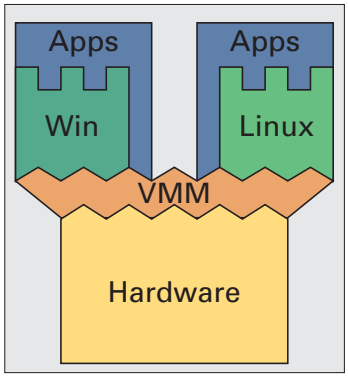

Abbildung 3: Der häufigste Fall eines klassischen Hypervisor, der direkt auf der Hardware läuft, deren ISA-Schnittstelle für seine Gäste repliziert und außerdem noch ihre privilegierten Operationen abfängt und stellvertretend erledigt.

Gäste in der Regel nichts davon wissen, dass sie die Hardware nicht wie erwartet exklusiv besitzen, sondern teilen müssen (**Abbildung 3**).

Im zweiten Fall unterscheiden sich die ISA von Gast und Hypervisor, Letzterer muss auch die Übersetzung der Gast-Befehle in die Sprache seiner Hardware übernehmen (Emulation). Das beherrschen unter Linux etwa Bochs **(2)** oder Qemu **(3)** (auf Qemu geht ein weiterer Beitrag in diesem Heft ausführlich ein).

Eine andere Gruppe aus dieser Klasse virtueller Maschinen – die Co-Designed Virtual Machines (CVM) – wird gemeinsam mit der Hardware entwickelt. Ein häufiges Ziel dabei ist es, neue Hardware mit einer verbreiteten und etablierten ISA kompatibel zu halten. Ein Beispiel für die Nützlichkeit liefert IBMs System/38, ein Vorläufer der AS/400 (heute I-Series), bei dem sich in den 90er Jahren dank dieser Technik die komplette Hardwareplattform transparent ersetzten ließ (IPMI-CPUs gegen PowerPC-Prozessoren).

Eine Welt

Die System-VMs mit gleicher ISA lassen sich weiter danach unterscheiden, wo der Virtualisierungslayer liegt, denn dafür gibt es verschiedene Möglichkeiten. Setzt er direkt auf der Hardware auf (**Abbildung 3**), spricht man von einem klassischen Hypervisor oder einer vollständigen Virtualisierung. Hier wäre etwa der ESX-Server von VMware einzuordnen (mehr über dieses Produkt in einem weiteren Beitrag). Befindet sich der Virtualisierungslayer dagegen oberhalb eines Host-Betriebssystems, heißt das Modell Hosted Virtualization (ein Beispiel zeigt **Abbildung 4**).

Der VMware Server **(4)** (ehemals GSX-Server), die Workstation aus gleichem Haus oder auch Parallels **(5)** stehen für diesen Typ, ebenso User Mode Linux **(6)** (dessen Schöpfer und Maintainer, Jeff Dike, einen Workshop für dieses Heft beigesteuert hat).

Dabei kann zusätzlich die ISA emuliert werden, was beispielsweise passiert, wenn Microsofts Virtual PC unter einem Mac-Betriebssystem Windows startet (**Abbildung 4**). Alternativ stützen sich alle Gäste gemeinsam auf den Betriebssystem-Kern des Hosts. Diese Technik heißt Betriebssystem-Virtualisierung. Vertreter dafür sind etwa die BSD Jails **(7)**, die Zonen in Solaris **(8)**, außerdem Virtuozzo **(9)** beziehungsweise dessen freier Ableger OpenVZ **(10)** oder das Vserver-Projekt **(11)** (mehr dazu bietet ein Workshop in diesem Heft).

Als Spezialfall des klassischen Hypervisor erlangte ein Modell große Beliebtheit, bei dem sich die Gäste im Klaren darüber sind, dass sie in einer virtuellen Umgebung laufen. Dafür müssen sie allerdings gepatcht werden, was nur für quelloffene Systeme einfach zu bewerkstelligen ist. Dann allerdings kann durch die Kooperation von Gast und Gastgeber bei der Behandlung sensitiver Operationen ein beachtlicher Performancevorteil erreicht werden. Dieser Fall heißt Paravirtualisierung und sein prominentester Vertreter heutzutage ist Xen **(12)**. (Mit Xen beschäftigen sich mehrere Beiträge in diesem Heft en detail.)

Von allen diesen Spielarten ist keine a priori die beste, stattdessen entsprechen ihnen unterschiedliche Einsatzszenarien. Emulation kostet zwar Performance, eröffnet aber die Möglichkeit, Software einer fremden Welt zum Laufen zu bringen. Paravirtualisierung ist schnell, verlangt dafür aber modifizierbare Gastbetriebssysteme, scheidet also etwa für Windows daher praktisch aus.

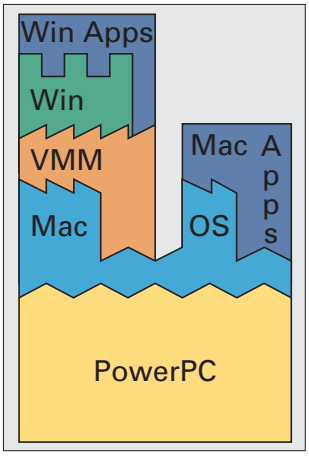

Abbildung 4: Eine System-VM, die innerhalb eines Betriebssystems ausgeführt wird und eine fremde ISA emuliert, hat dagegen diese, schon etwas kompliziertere Architektur.

Die Virtualisierung des Betriebssystems ist besonders Ressourcen-schonend und daher beispielsweise bei Providern auch besonders beliebt, sie beschränkt die Gäste allerdings auf das Operating System ihres Hosts.

Die Hardware-Unterstützung ist zwar effektiv, aber noch nicht mit jeder CPU und jeder Virtualisierungslösung nutzbar.

Infos

(1) Robert P. Goldberg, „Architectural Principles for Virtual Computer Systems": PhD-Thesis, Harvard University, 1972
(2) Bochs: (http://bochs.sourceforge.net)
(3) Qemu-Homepage: (http://www.qemu.com)
(4) VMware Server: (http://www.vmware.com)
(5) Parallels: (http://www.parallels.com)
(6) UML: (http://user-mode-linux.sourceforge.net)
(7) BSD Jails: (http://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/jail.html)
(8) Solaris Zones: (http://docs.sun.com/app/docs/doc/819-4323?l=de&a=load)
(9) Virtuozzo: (http://www.swsoft.com)
(10) OpenVZ: (http://openvz.org)
(11) Vserver: (http://linux-vserver.org/Overview)
(12) Xen: (http://www.cl.cam.ac.uk/Research/SRG/netos/xen/)
(13) Intel VT: (http://cache-www.intel.com/cd/00/00/23/66/236644_236644.pdf)
(14) AMD IOMMU: (http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf)

Hardwarehilfe

Die x86-Architektur war lange Zeit nicht besonders virtualisierungsfreundlich. Um mit einem Hypervisor kooperieren zu können, müssen sich die Gäste einer Oberaufsicht unterwerfen, die sie aus ihrem Dasein als Solo-Betriebssystem nicht kennen. Alle privilegierten Befehle, die im so genannten Ring 0 der CPU laufen und dem Betriebssystem vorbehalten sind, müssen jetzt eine Exception auslösen. Denn im Virtualisierungsfall besetzt der Hypervisor die Ebene der höchsten Berechtigungen.

Er würde an einer solchen Unterbrechung erkennen, dass ein jetzt minder privilegiertes Gastbetriebssystem einen Zugriff versucht, der ihm nicht mehr gestattet werden darf. So könnte er eingreifen, die Instruktion abfangen, ihre Ausführung emulieren und das Resultat an den Gast zurückgeben.

Ältere x86-CPUs lösen allerdings nicht in jedem Fall die erforderliche Exception aus. Auch eine Herabstufung einzelner Instruktionen für den Gast (De-Privileging) ist nicht möglich, denn einige Maschinenbefehle haben im Userlevel eine andere Semantik.

Ein Workaround, den beispielsweise VMware verwendet, besteht darin, den gesamten Code, den eine VM auszuführen plant, vorher auf diese kritischen Instruktionen hin zu durchforsten (Prescan) und die Fundstellen dann zu modifizieren (Binary Translation). Selbstverständlich kostet das Performance. Einem ähnlichen Zweck dient auch das Patchen des Gastes à la Xen, was zwar performanter ist, aber modifizierbare Gäste erfordert.

Seit kurzem unterstützen nun auch die CPU-Hersteller der x86-Welt, Intel und AMD, aktiv die Virtualisierung. Intel mit seiner Intel Virtualization Technology (VT, Codename Vanderpool, (13)), AMD mit der AMD I/O Virtualization Technology (Codename Pacifica, (14)).

Die Lösung von AMD implementiert spezielle Prozessorfunktionen, die den Hypervisor bei der Adressenumsetzung unterstützen und in seinem Auftrag außerdem kontrollieren können, ob bestimmte Zugriffe eines Gastsystems auf Geräte überhaupt erlaubt werden dürfen. (Mehr dazu in einem weiteren Beitrag in diesem Heft.)

Intels VT – das viele neuere Prozessoren bereits mitbringen – bietet ebenfalls bestimmte Befehlserweiterungen des Prozessors (Virtual Machine Extensions). Sie erlauben es, die bekannte Hierarchie von CPU-Operationen nach dem erwähnten Modell einer Hierarchie verschieden privilegierter Ringe um zwei neue Modi zu ergänzen: Root- beziehungsweise Non-Root-Modus.

Diese Modi kann der Hypervisor steuern, der selber im Root-Modus läuft. Er kann das Gast-Betriebssystem in den Non-Root-Modus versetzen, indem es gefahrlos Ring-0-Instruktionen verwenden darf. Dabei bemerkt es selber keinen Unterschied zu den Abläufen, die es als Alleinherrscher eines Rechners kennt.

Xen 3.0 unterstützt sowohl AMDs wie auch Intels Technologie und kann mit VT-Unterstützung auch Windows ohne Änderung neben para-virtualisierten Gästen ausführen.

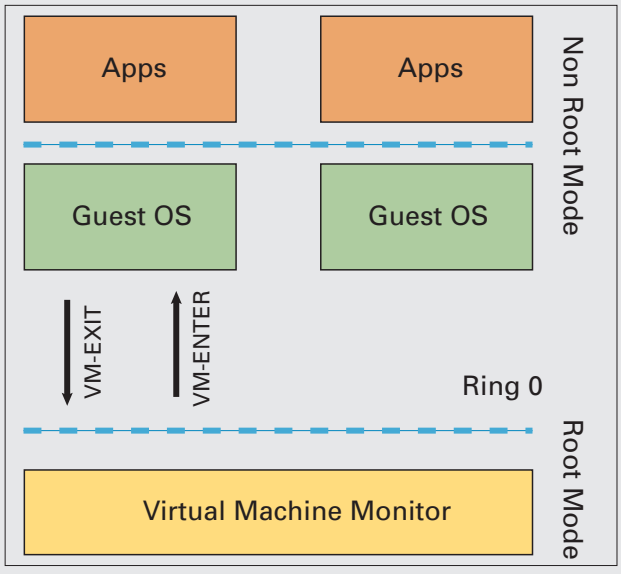

Abbildung 5: Intels VT erlaubt es Gastbetriebssystemen, wie gewohnt Ring 0 zu verwenden, derweil der VMM in einem besonders privilegierten Root-Modus läuft.