

Title: Baumgarten: GBS (24.01.2014)

Date: Fri Jan 24 08:33:31 CET 2014

Duration: 63:02 min

Pages: 22

Grundlagen: Betriebssysteme und Systemsoftware (GBS)

Uwe Baumgarten

1. Übersicht

- Ziele der Vorlesung
- Motivation
- Themen der Vorlesung
- Literaturübersicht

Quelle: [JS12] Kap. 1

The screenshot shows a presentation slide titled "Exkurs: Strukturierungskonzepte" (Excursion: Structuring Concepts) from a PDF document. The slide content includes:

- Konzepte**
 - Prozedur-orientiert vs. Nachrichten-orientiert
 - Kerne, Mikro-, Nano-, Pico-, Exo-Kerne
 - Virtualisierung
- BS-Strukturen**
 - Monolithisch
 - Systemken
 - Schichtung
 - Problem: „Up-calls“
 - Hierarchie abstrakter Maschinen

Handwritten annotations in green and blue ink are present on the slide. A vertical red line on the left side is labeled "10 Entwurf von Betriebssystemen".

At the bottom of the slide, there is a footer with the following text:

James E. Smith
University of Wisconsin-Madison

Ravi Nair
virtualization has become an important tool in computer system design, and virtual machines are used in a number of disciplines ranging from operating systems to programming languages to processor architectures. By freeing developers and

A computer's instruction set architecture (ISA) clearly exemplifies the advantages of well-defined interfaces. Well-defined interfaces permit development of interacting computer subsystems not only in different organizations but also at different times, sometimes years apart. For example, Intel and

The screenshot also shows the presentation software interface, including a menu bar, a toolbar, and a status bar at the bottom with the date "Fr, 24. Jan, 08:34".

Exkurs: Virtualisierung (2)

- Grundlagen
 - Smith, Nair: The Architecture of Virtual Machines, IEEE Computer, 2005
http://www.itc.ku.edu/~kulkarni/teaching/archieve/EECS800-Spring-2008/smith_nair.pdf
- Quellen
 - Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2
<http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf>
 - Virtualisierungsfunktion Intel VT-x aktivieren
http://www.thomas-krenn.com/de/wiki/Virtualisierungsfunktion_Intel_VT-x_aktivieren

IEEE-Computer-2005-smith_nair.pdf

Datei Bearbeiten Ansicht Gehe zu Lesezeichen Hilfe

Vorherige Nächste 1 (1 von 7) Auf Seitenbreite einpassen

COVER FEATURE

The Architecture of Virtual Machines

A virtual machine can support individual processes or a complete system depending on the abstraction level where virtualization occurs. Some VMs support flexible hardware usage and software isolation, while others translate from one instruction set to another.

James E. Smith
University of Wisconsin-Madison

Ravi Nair
IBM T.J. Watson Research Center

Virtualization has become an important tool in computer system design, and virtual machines are used in a number of disciplines ranging from operating systems to programming languages to processor architectures. By freeing developers and users from traditional interface and resource constraints, VMs enhance software interoperability, system impregnability, and platform versatility. Because VMs are the product of diverse groups

A computer's instruction set architecture (ISA) clearly exemplifies the advantages of well-defined interfaces. Well-defined interfaces permit development of interacting computer subsystems not only in different organizations but also at different times, sometimes years apart. For example, Intel and AMD designers develop microprocessors that implement the Intel IA-32 (x86) instruction set, while Microsoft developers write software that is compiled to the same instruction set. Because both

32

Computer Published by the IEEE Computer Society 0010-9126/05/02.00 © 2005 IEEE

Menu TUM-GBS-Fol... IEEE-Comput... Fr. 24. Jan. 08:47

IEEE-Computer-2005-smith_nair.pdf

TUM-GBS-Fol-2013-ALLE-1-233.pdf — TUM-GBS-Fol-2013.ppt [Kompatibilitätsmodus]

Technische Universität München

Exkurs: Virtualisierung (2)

- Grundlagen
 - Smith, Nair: The Architecture of Virtual Machines, IEEE Computer, 2005
http://www.itc.ku.edu/~kulkarni/teaching/archieve/EECS800-Spring-2008/smith_nair.pdf
- Quellen
 - Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2
<http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf>
 - Virtualisierungsfunktion Intel VT-x aktivieren

James E. Smith
University of Wisconsin-Madison

Ravi Nair
IBM T.J. Watson Research Center

Virtualization has become an important tool in computer system design, and virtual machines are used in a number of disciplines ranging from operating systems to programming languages to processor architectures. By freeing developers and

A computer's instruction set architecture (ISA) clearly exemplifies the advantages of well-defined interfaces. Well-defined interfaces permit development of interacting computer subsystems not only in different organizations but also at different times, sometimes years apart. For example, Intel and

Menu TUM-GBS-Fol... IEEE-Comput... Fr. 24. Jan. 08:47

IEEE-Computer-2005-smith_nair.pdf

Datei Bearbeiten Ansicht Gehe zu Lesezeichen Hilfe

Vorherige Nächste 1 (1 von 7) Auf Seitenbreite einpassen

with different goals, however, there has been relatively little unification of VM concepts. Consequently, it is useful to take a step back, consider the variety of VM architectures, and describe them in a unified way, putting both the notion of virtualization and the types of VMs in perspective.

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 programmers can then create, write, and read files without knowing the hard disk's construction and physical organization.

groups satisfy the ISA specification, the software can be expected to execute correctly on any PC built with an IA-32 microprocessor.

Unfortunately, well-defined interfaces also have their limitations. Subsystems and components designed to specifications for one interface will not work with those designed for another. For example, application programs, when distributed as compiled binaries, are tied to a specific ISA and depend on a specific operating system interface. This lack of interoperability can be confining, especially in a world of networked computers where it is advantageous to move software as freely as data.

Virtualization provides a way of getting around such constraints. Virtualizing a system or component—such as a processor, memory, or an I/O device—at a given abstraction level maps its interface and visible resources onto the interface and resources of an underlying, possibly different, real system. Consequently, the *real system* appears as a different virtual system or even as multiple virtual systems.

Unlike abstraction, virtualization does not necessarily aim to simplify or hide details. For example, in Figure 1b, virtualization transforms a single large disk into two smaller virtual disks, each of which

32

Computer Published by the IEEE Computer Society 0010-9126/05/02.00 © 2005 IEEE

Menu TUM-GBS-Fol... IEEE-Comput... Fr. 24. Jan. 08:47

appears to have its own tracks and sectors. Virtualizing 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.

VIRTUAL MACHINES

The concept of virtualization can be applied not only to subsystems such as disks but to an entire machine. To implement a virtual machine, developers add a software layer to a real machine to support the desired architecture. By doing so, a VM can circumvent real machine compatibility and hardware resource constraints.

Architected interfaces

A discussion of VMs is also a discussion about computer architecture in the pure sense of the term. Because VM implementations lie at architected interfaces, a major consideration in the construction of a VM is the fidelity with which it implements these interfaces.

Architecture, as applied to computer systems, refers to a formal specification of an interface in the system, including the logical behavior of resources managed via the interface. **Implementation** describes the actual embodiment of an architecture. Abstraction levels correspond to implementation layers, whether in hardware or software, each associated with its own interface or architecture.

Figure 2 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.

Instruction set architecture. The ISA marks the division between hardware and software, and consists of interfaces 3 and 4 in Figure 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).

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

Figure 1 consists of two diagrams, (a) and (b), illustrating disk storage abstraction and virtualization. Diagram (a) shows a real disk with a file being written to it. A green arrow labeled 'Abstraction' points to a simplified interface where the file is represented as a single block. Diagram (b) shows a virtual disk with a file being written to it. A green arrow labeled 'Virtualization' points to a different interface where the file is represented as a single block, but the underlying resources are different from the real disk.

Figure 2. Computer system architecture. Key implementation layers communicate vertically via the instruction set architecture (ISA), application binary interface (ABI), and application programming interface (API).

Figure 2 is a layered diagram of computer system architecture. The layers from top to bottom are: Software (Application programs, Libraries, Operating system), Execution hardware, System interconnect (bus), and Hardware (I/O devices and networking, Main memory). Interfaces 1, 2, 3, and 4 are marked between the layers. Interface 1 is between Application programs and Libraries. Interface 2 is between Libraries and Operating system. Interface 3 is between Operating system and Execution hardware. Interface 4 is between Execution hardware and System interconnect (bus). The Hardware layer is further divided into I/O devices and networking and Main memory.

Architected interfaces

A discussion of VMs is also a discussion about computer architecture in the pure sense of the term. Because VM implementations lie at architected interfaces, a major consideration in the construction of a VM is the fidelity with which it implements these interfaces.

Architecture, as applied to computer systems, refers to a formal specification of an interface in the system, including the logical behavior of resources managed via the interface. **Implementation** describes the actual embodiment of an architecture. Abstraction levels correspond to implementation layers, whether in hardware or software, each associated with its own interface or architecture.

Figure 2 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.

Instruction set architecture. The ISA marks the division between hardware and software, and consists of interfaces 3 and 4 in Figure 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).

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

Figure 2 is a layered diagram of computer system architecture. The layers from top to bottom are: Software (Application programs, Libraries, Operating system), Execution hardware, System interconnect (bus), and Hardware (I/O devices and networking, Main memory). Interfaces 1, 2, 3, and 4 are marked between the layers. Interface 1 is between Application programs and Libraries. Interface 2 is between Libraries and Operating system. Interface 3 is between Operating system and Execution hardware. Interface 4 is between Execution hardware and System interconnect (bus). The Hardware layer is further divided into I/O devices and networking and Main memory.


Exkurs: Virtualisierung (2)

- Grundlagen
 - Smith, Nair: The Architecture of Virtual Machines, IEEE Computer, 2005 http://www.itc.ku.edu/~kulkarni/teaching/archieve/EECS800-Spring-2008/smith_nair.pdf
- Quellen
 - Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2 <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf>
 - Virtualisierungsfunktion Intel VT-x aktivieren http://www.thomas-krenn.com/de/wiki/Virtualisierungsfunktion_Intel_VT-x_aktivieren

Exkurs: Virtualisierung (3)

- Intel Virtualization Technology (Intel VT) (Vanderpool)
- Hardware-Unterstützung durch das Host-System
- SVM (Secure VM) als effiziente VM durch Befehlssatzerweiterung
 - Befehlsausführung nicht emuliert, sondern direkt durch den Host ausgeführt
- Linux
 - proc/cpuinfo
- Paravirtualisierung
 - Steigerung der Ausführungsgeschwindigkeit durch Anpassungen an die Virtualisierungsschnittstelle

IEEE-Computer-2005-smith_nair.pdf
TUM-GBS-Fol-2013-ALLE-1-233.pdf — TUM-GBS-Fol-2013.ppt [Kompatibilitätsmodus]

Technische Universität München 


Exkurs: Virtualisierung (3)

- Intel Virtualization Technology (Intel VT) (Vanderpool)
- Hardware-Unterstützung durch das Host-System
- SVM (Secure VM) als effiziente VM durch Befehlssatzerweiterung
 - Befehlsausführung nicht emuliert, sondern direkt durch den Host ausgeführt
- Linux
 - proc/cpuinfo

by using the virtualization technology, the virtualizing software in a system VM is typically referred to as the *virtual machine monitor* (VMM). Figure 3 depicts process and system VMs, with compatible interfaces illustrated graphically as meshing boundaries. In a process VM, the virtualizing software is at the ABI or API level, atop the OS/HW combination. The runtime emulates both user-level instructions and either operating system (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.

Process and system VMs


Menu TUM-GBS-Fol... IEEE-Comput... Fr. 24. Jan. 09:04

Technische Universität München 

SCSI – Einsatz - RAID

- Plattenarrays (RAID) (Redundant Array of Inexpensive Disks)
 - Virtuelle/logische Platte
 - RAID 0: Striping der Blöcke
 - RAID 1: Spiegelung
 - RAID 0+1: Striping mit Spiegelung
 - RAID 3: Bit-Level-Striping + separate Parity-Platte
 - RAID 4:
 - RAID 5: Block-Level-Striping + verteilte Parity-Blöcke
 - RAID 6:


© UB TUM GBS WS 2013/14 Grundlagen: Betriebssysteme und Systemsoftware (IN0009) 226

Technische Universität München 

SCSI - Einführung

- SCSI = Small Computer Systems Interface
- Ziele: Leistungsfähige Anbindung von E/A-Geräten (universell)
- Einsatz:
 - Zunächst: Apple Macintosh + Sun Workstations
 - Jetzt: Hochleistungsworkstations + Server + RAID
- 1979 (SASI, Shugart) – 1982 (NCR) – 1986 (ANSI SCSI-1)
- Quellen
 - Niklaus Wirth, Jürg Gutknecht
Project Oberon: The Design of an Operating System and Compiler, Addison-Wesley, ACM Press, 1992
 - ANSI Working Draft:
„SCSI Architecture Model – 4“

© UB TUM GBS WS 2013/14 Grundlagen: Betriebssysteme und Systemsoftware (IN0009) 225

Technische Universität München 

SCSI – Familie und Leistung

- SCSI-Familie (viele teils verschiedene S.)
 - SCSI-1, SCSI-2, SCSI-3, Ultra-2, Ultra-3, Ultra-320, Ultra-640, iSCSI, Serial SCSI
- Leistung

Schnittstelle	Breite	Taktung	Durchsatz (max)	Länge (max)	Geräte Anzahl
SCSI	8	5 MHz	5 MB/s	6 m	8
Ultra SCSI	8	20 MHz	20 MB/s	1.5-3 m	8
Ultra-320	16	80 MHz DDR	320 MB/s	12 m	16
iSCSI					

© UB TUM GBS WS 2013/14 Grundlagen: Betriebssysteme und Systemsoftware (IN0009) 227

SCSI – Hardware

- SCSI-Bus
 - Datenleitungen
 - 8x
 - Parität
 - 1x
 - Steuerleitungen
 - 9x
 - SEL, BSY, REQ, ACK, C/D, MSG, I/O, RST, ATN
- Register
 - Write register
 - ODR, ICR, MR2, TCR
 - Read ports und register
 - CSD, CSB, BSR

SCSI – Software

- Rollen bei der Verarbeitung
 - Master
 - Server
- Einzelne Verarbeitung (Lesen, Schreiben)
 - Handshake für asynchrone Byte-Übertragung: M2S
 - Master: REQ=1
 - Master: Daten liegen an
 - Server: Lesen der Daten
 - Server: ACK=1
 - Master: REQ=0
 - Server: ACK=0
 - S2M
 - Master: REQ=1
 - Server: ACK=1
 - Server: Daten liegen an
 - Master: Lesen der Daten
 - Master: REQ=0
 - Server: ACK=0

SCSI – Phasen der Verarbeitung

- Selection phase (nur ein Initiator)
 - Initiator: SEL=1 + 1 of 8 Data lines
 - Target: BSY
 - Nach Auswahl: Target = Master, Initiator = Server
- Command phase
 - Target: Fordert Kommando an BSY=C/D=1
 - Initiator: Sendet Kommando
 - Übertragung des Kommandos (OP-Code)
- Data phase
 - C/D=MSG=0
 - REQ abwarten, I/O=0/1
- Status phase
 - BSY=C/D=1 MSG=0
 - Target: Sendet Statusinformation
- Message phase
 - BSY=MSG=C/D=1
 - Target: Sendet Message

SCSI – Phasen der Verarb. (2)

- Arbitration phase
 - Mehrere Initiatoren möglich
 - Erster Initiator startet:
 - BSY=1 und InitiatorDataLine=1
 - Arbitration Delay abwarten (200 ns)
 - Wenn kein „höherer“, dann „gewonnen“.
 - Falls doch: BSY=0
- Kommunikation erfolgt, wenn
 - Initiator und Target in den richtigen Phasen
 - Ansonsten: Mismatch BSR(PHM)=1

SCSI – Commandos & Adressierung

- Struktur
 - 1 Byte OP-Code
 - 5 und mehr Bytes Parameter
- Beispiel: Target = Disk Controller
 - Par1: Sektoradresse (3 Bytes)
 - Par2: Anzahl der Plattensektoren (1 Byte)
 - Par3: Plattenspez. Info (1 Byte)
- Adressierung
 - Hostadapter: 7
 - Festplatte: 0
 - CD-Rom: 2
- Basis: Datenleitungen

SCSI – Software

- Rollen bei der Verarbeitung
 - Master
 - Server
- Einzelne Verarbeitung (Lesen, Schreiben)
 - Handshake für asynchrone Byte-Übertragung: M2S
 - Master: REQ=1
 - Master: Daten liegen an
 - Server: Lesen der Daten
 - Server: ACK=1
 - Master: REQ=0
 - Server: ACK=0
 - S2M
 - Master: REQ=1
 - Server: ACK=1
 - Server: Daten liegen an
 - Master: Lesen der Daten
 - Master: REQ=0
 - Server: ACK=0

SCSI - Einführung

- SCSI = Small Computer Systems Interface
- Ziele: Leistungsfähige Anbindung von E/A-Geräten (universell)
- Einsatz:
 - Zunächst: Apple Macintosh + Sun Workstations
 - Jetzt: Hochleistungsworkstations + Server + RAID
- 1979 (SASI, Shugart) – 1982 (NCR) – 1986 (ANSI SCSI-1)
- Quellen
 - Niklaus Wirth, Jürg Gutknecht
Project Oberon: The Design of an Operating System and Compiler, Addison-Wesley, ACM Press, 1992
 - ANSI Working Draft:
„SCSI Architecture Model – 4“