Script **generated by TTT**

Title: Baumgarten: GBS (24.01.2014)

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

Duration: 63:02 min

Pages: 22

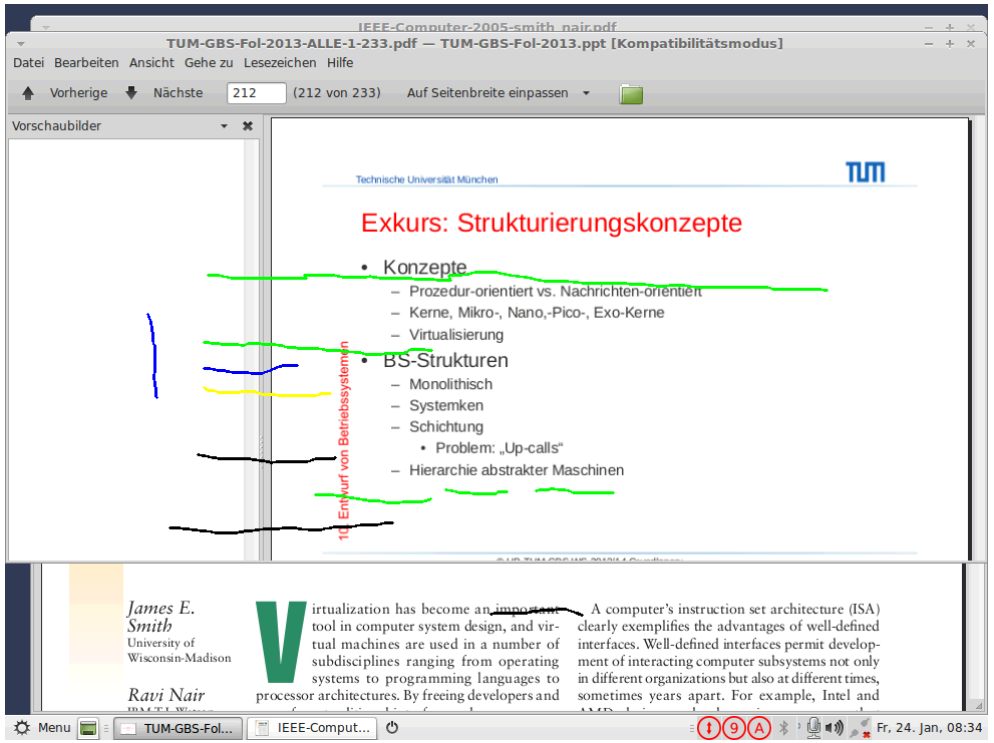
Grundlagen: Betriebssysteme und Systemsoftware (GBS)

Uwe Baumgarten

1. Übersicht

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

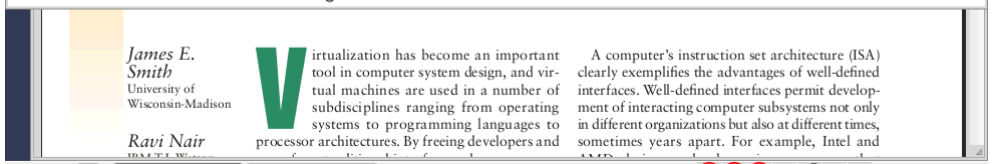
Quelle: [JS12] Kap. 1



Exkurs: Virtualisierung (2)

- Grundlagen
  - Smith, Nair: The Architecture of Virtual Machines, IEEE Computer, 2005
    http://www.ittc.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
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.ittc.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



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 subdisciplines 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 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.

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 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.

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.

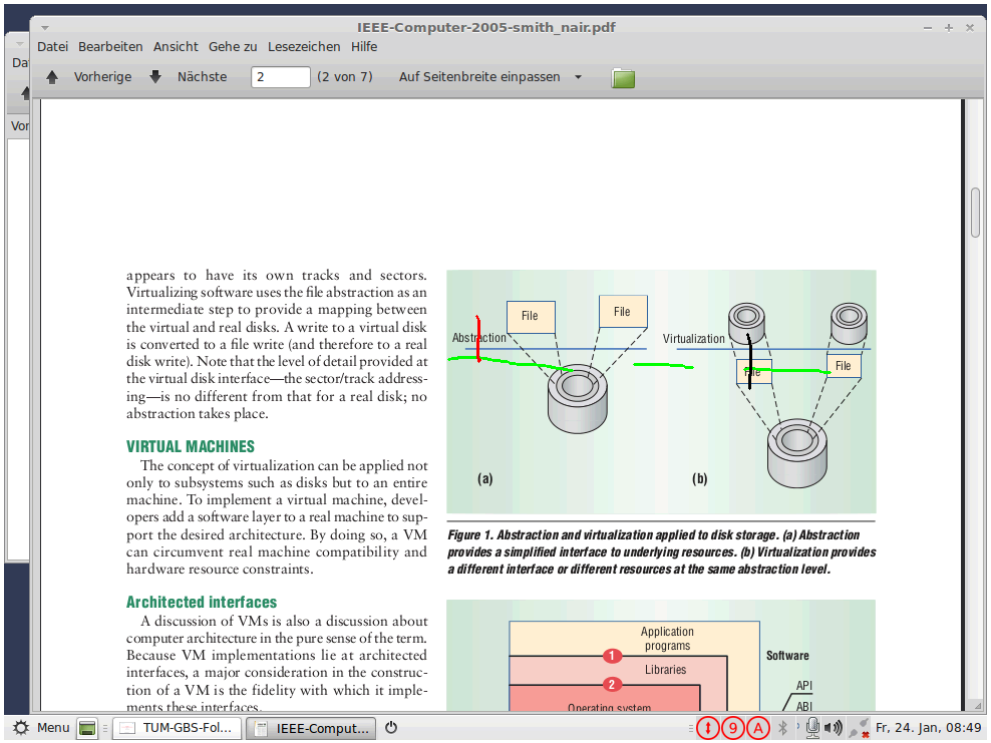
Figure 1a shows an example of abstraction applied to disk storage. The operating system abstracts hard-disk addressing details—for example, that it is comprised of sectors and tracks—so that the disk appears to application software as a set of variable-sized files. Application programmers can then create, write, and read files without knowing the hard disk's construction and physical organization.

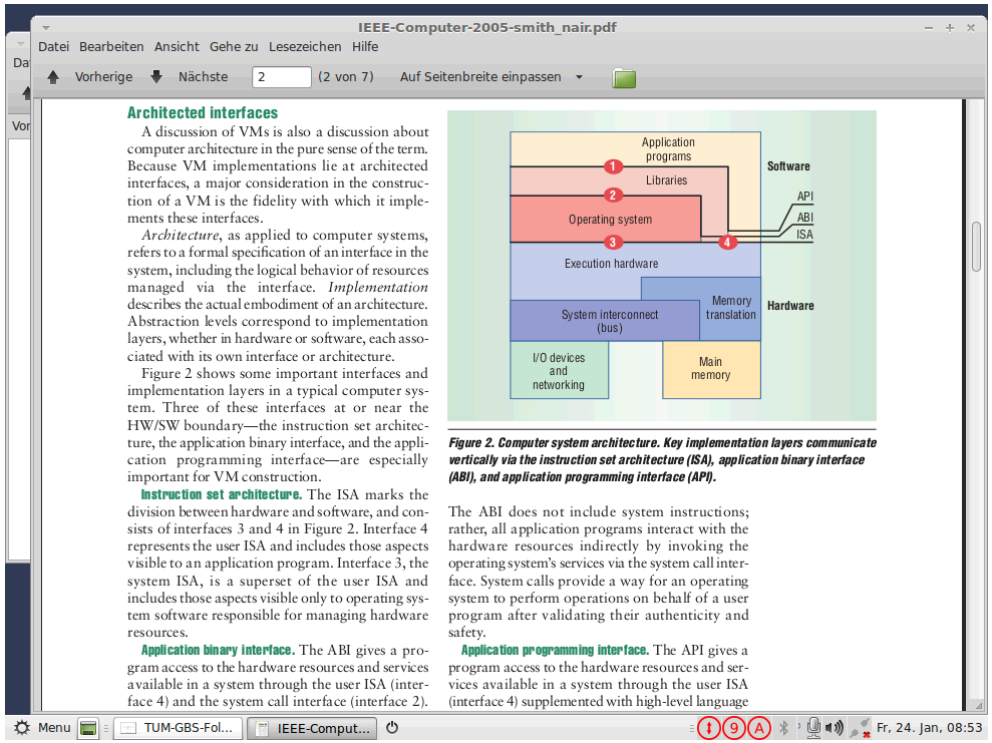
Virtualization provides a way of getting around such constraints. Virtualizing a system or component—such as a processor, memory, or an I/O device—at a given abstraction level maps its interface and visible resources onto the interface and resources of an underlying, possibly different, real system.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

Published by the IEEE Computer Society
0018-9162/05/$20.00 © 2005 IEEE

Menu TUM-GBS-Fol... IEEE-Comput... Fr, 24. Jan, 08:47
Menu TUM-GBS-Fol... IEEE-Comput... Fr, 24. Jan, 08:48





Exkurs: Virtualisierung (2)

- Grundlagen
  - Smith, Nair: The Architecture of Virtual Machines, IEEE Computer, 2005 http://www.ittc.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

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

(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

ing 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

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“

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:

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					

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“