



Betriebssysteme und nebenläufige Programmierung

SoSe 2026

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 20. März 2026



Betriebssysteme und nebenläufige Programmierung

SoSe 2026

11 Virtualisierung

- ➔ Virtuelle Prozessoren: Emulator, z.B. QEMU, JVM
- ➔ Virtuelle Prozessumgebungen: normales BS
- ➔ Virtuelles BS: ABI für BS X auf BS Y , z.B. WINE
- ➔ Virtueller Desktop: z.B. RDP, VNC
- ➔ Virtuelle Ressourcen
 - ➔ z.B. *Storage Area Network* (SAN), virtuelle Netzwerke
- ➔ Virtuelles Laufzeitsystem (Sandboxing)
 - ➔ *Container*
- ➔ Virtuelle Computer (virtuelle Maschine)
 - ➔ wirkt (auch für BS) wie realer Computer

Anmerkungen zu Folie 455:

- ➔ ABI = *Application Binary Interface*, also binäre Aufrufchnittstelle eines Software-Moduls, hier: des Betriebssystems
- ➔ Zum Nachlesen:
 - ➔ Eduard Glatz. *Betriebssysteme – Grundlagen, Konzepte, Systemprogrammierung*. 3. Auflage, dpunkt.verlag, 2015. Kap 12.1 und 12.2

Motivation

- ➔ Server (WWW, Mail, ...) laufen i.a. auf eigenen Rechnern
 - sicherer und zuverlässiger
 - unterschiedliche Anforderungen an Systemsoftware / BS
- ➔ Statt einzelner Rechner: virtuelle Maschinen auf einem Rechner
 - kostengünstiger / energiesparender
 - Migration virtueller Maschinen (VMs) zwischen Rechnern möglich
 - Erneuerung der Hardware problemlos(er)
- ➔ Realisierung durch *Hypervisor* (*Virtual Machine Monitor*, VMM)
 - spielt BS auf VM vor, dass es direkt auf der Hardware läuft
 - VMs sind logisch getrennt; ggf. auf jeder VM anderes BS

Anmerkungen zu Folie 456:

Zum Nachlesen:

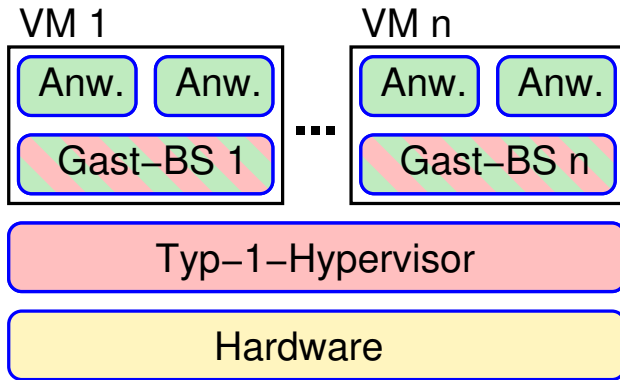
- ➔ Andrew S. Tanenbaum, Herbert Bos. *Moderne Betriebssysteme*. 4. Auflage. Pearson, 2016. Kap. 7
- ➔ Eduard Glatz. *Betriebssysteme – Grundlagen, Konzepte, Systemprogrammierung*. 3. Auflage, dpunkt.verlag, 2015. Kap 12.3
- ➔ William Stallings. *Operating Systems – Internals and Design Principles*. 8. Auflage. Pearson, 2015. Kap. 14



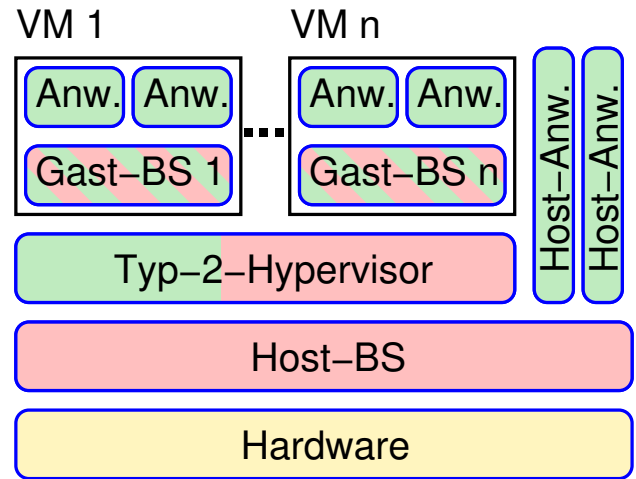
(Animierte Folie)

Schichtenmodell

Typ-1-Hypervisor



Typ-2-Hypervisor



Systemmodus
 Benutzermodus
 Virtueller Systemmodus

Mischform: Hypervisor im Host-BS integriert

Anmerkungen zu Folie 457:

- ➔ Das Gast-Betriebssystem muss in einem „virtuellen Systemmodus“ laufen, d.h., es läuft tatsächlich im Benutzermodus, kann aber trotzdem (mit Hilfe des Hypervisors) privilegierte Befehle ausführen.
- ➔ Ein Typ-2-Hypervisor setzt auf dem Host-Betriebssystem auf, kann also dessen Dienste für die Ein-/Ausgabe nutzen. Da der Typ-2-Hypervisor aber das Multiplexing von CPU und Speicher zwischen den VMs realisieren muß, muß zumindest ein Teil des Typ-2-Hypervisors im Systemmodus laufen.



Beispiele

- ➔ Typ-I-Hypervisor
 - ➔ VMware ESX
- ➔ Typ-II-Hypervisor
 - ➔ VMware Workstation, VirtualBox, Xen
 - ➔ Xen kann durch eigenes Linux auch direkt auf HW aufsetzen
- ➔ In BS integriert:
 - ➔ KVM (Linux), Hyper-V (Windows)



Realisierung

- ➔ Gast-BS muss im Benutzermodus ausgeführt werden
 - ➔ Problem: Ausführung **sensitiver Befehle**
 - ➔ Verhalten im Benutzer- und Systemmodus unterschiedlich
- ➔ Sensitive Befehle müssen vom Hypervisor ausgeführt (emuliert) werden
- ➔ Einfach, falls alle sensitiven Befehle auch privilegiert sind
 - ➔ dann: Trap zum Hypervisor, dort Emulation
- ➔ Bei vielen (älteren) Prozessoren nicht der Fall (z.B. Pentium)
 - ➔ Lösung: Binärübersetzung des Codes
 - ➔ Alternative: **Paravirtualisierung**
 - ➔ Gast-BS wird angepasst

Anmerkungen zu Folie 459:

- ➔ Beim Intel Pentium gibt es beispielsweise den Befehl POPF, der das Flag-Register mit dem obersten Kellerelement überschreibt. Im Flag-Register gibt aber ein Bit an, ob Interrupts erlaubt oder gesperrt sind. Da Code im Benutzermodus nicht die Interrupts sperren darf, wird dieses Bit im Benutzermodus einfach ignoriert (statt eine Exception zu erzeugen). Damit würde ein Gast-BS, das im Benutzermodus läuft, aber nicht mehr funktionieren, da der Hypervisor die Ausführung des POPF Befehls nicht mitbekommt.
- ➔ Bei der Paravirtualisierung werden alle sensitiven Befehle im Gast-BS durch explizite Aufrufe des Hypervisors (*Hypercalls*) ersetzt.
Dabei kann idealerweise eine Hypervisor-unabhängige Schnittstelle verwendet werden. Eine der Realisierungen dieser Schnittstelle setzt dann die Hypercalls direkt in (privilegierte) Befehle um, so daß das Betriebssystem auch direkt auf der Hardware aufsetzen kann.
- ➔ Moderne Prozessoren mit „Virtualisierungstechnologie“ stellen sicher, dass alle sensitiven Befehle auch privilegiert sind, der Befehlssatz also virtualisierbar ist. Zusätzlich bieten sie weitere Unterstützung für die Virtualisierung, siehe später.

459-1

11.2 Virtuelle Maschinen ...



Binärübersetzung

- ➔ Idee: Code, der im Gastsystem im Systemmodus laufen muß, wird zur Laufzeit ersetzt
 - ➔ der neue Code beinhaltet keine sensitiven Befehle mehr
- ➔ Ersetzung erfolgt bei Bedarf auf Ebene von Basisblöcken
 - ➔ Basisblock: lineare Befehlsfolge, höchstens ein Sprungziel am Anfang, höchstens ein (i.a. bedingter) Sprung am Ende
 - ➔ Hypervisor hält einen Cache mit bereits übersetzten Basisblöcken
 - ➔ am Ende eines Blocks: Rückkehr zum Hypervisor
- ➔ Wird aus *Performance*-Gründen z.T. auch bei virtualisierbaren Prozessoren eingesetzt

Anmerkungen zu Folie 460:

- ➔ Code im Benutzermodus ist in der Regel unkritisch und muß daher nicht übersetzt werden.
- ➔ Der neue Code ist unter dieser Voraussetzung nahezu identisch zum ursprünglichen Code. Lediglich sensitive Befehle werden durch andere Befehle ersetzt, z.B. durch einen Trap zum Hypervisor.
- ➔ Am Ende springt der neue Basisblock wieder zum Hypervisor, der den Folgeblock bestimmt und diesen ggf. übersetzt, falls noch kein entsprechender Eintrag im Cache liegt.
Dieser Sprung in den Hypervisor am Ende eines Basisblocks kann entfallen, wenn alle möglichen Nachfolgeblöcke bereits übersetzt wurden. In diesem Fall kann ein direkter Sprung in den Nachfolgeblock erfolgen.
- ➔ Da die Ausführung eines Trap-Befehls recht aufwendig ist, wird die Technik der Binärübersetzung auch bei virtualisierbaren Prozessoren verwendet, da sie teilweise eine schnellere Ausführung ermöglicht.

460-1

11.2 Virtuelle Maschinen ...

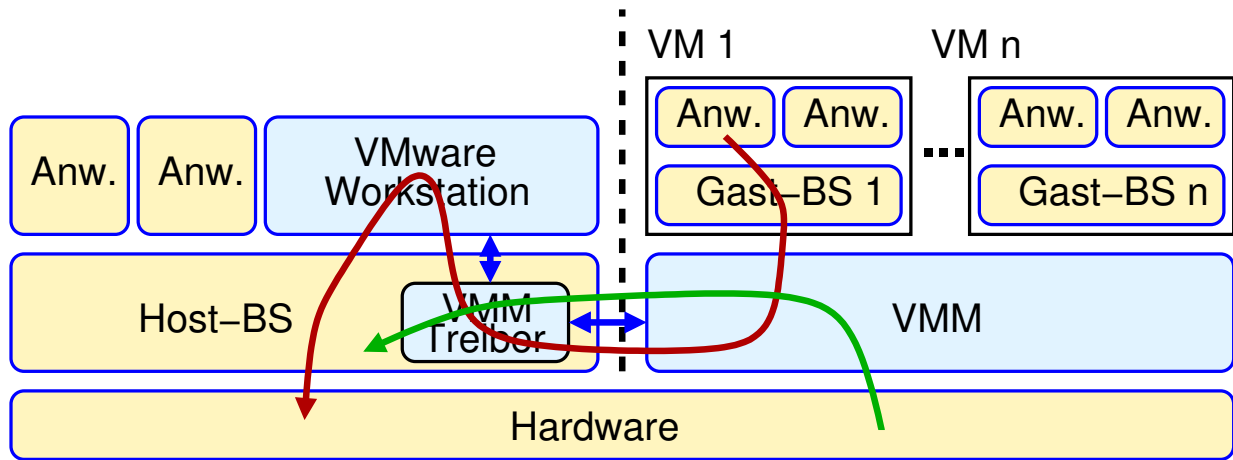


Typ-2-Hypervisor

- ➔ Typ-2-Hypervisor ist „normale“ Anwendung
 - ➔ kann so aber CPU und Speicher nicht multiplexen
 - ➔ Hypervisor muss teilweise im Systemmodus laufen
- ➔ Daher: Aufspaltung in Anwendung, Gerätetreiber und VMM
- ➔ Anwendung: Bedienoberfläche, Realisierung von E/A über Host-BS
- ➔ Gerätetreiber: ermöglicht Ausführung des VMM im Systemmodus, emuliert Interrupts
 - ➔ *World-Switch*: Umschaltung zwischen Host-BS und VMM
- ➔ VMM: Multiplexing von CPU und Speicher, Binärübersetzung, Unterbrechungsbehandlung

(Animierte Folie)

Typ-2-Hypervisor: Beispiel VMware



Routing einer E/A-Anfrage

Behandlung Interrupt, falls VMM läuft

- ➔ Ausgeführt wird entweder Host-BS oder VMM
- ➔ *World-Switch* durch VMM-Treiber sichert Zustand des Host-BS

Anmerkungen zu Folie 462:

Mehr Informationen zu VMware finden Sie in den folgenden Vortragsfolien:

- ➔ <http://download3.vmware.com/vmworld/2005/pac097.pdf>

Speichervirtualisierung

- ➔ Problem: jedes Gast-BS erstellt Seitentabellen unabhängig
 - ➔ dabei ggf. Abbildung auf dieselben Kacheln
- ➔ Hypervisor muß die Kacheln ggf. ändern
 - ➔ dazu: Schattentabellen im Hypervisor
- ➔ Problem: Hypervisor bekommt nur das Laden der Seitentabelle in die MMU mit, nicht aber spätere Veränderungen
- ➔ Mögliche Lösungen:
 - ➔ Seiten der Seitentabelle schreibschützen
 - ➔ Hinzufügen neuer Kacheln erst bei Seitenfehler
 - ➔ Löschen von Seiten mit privilegiertem Befehl (wegen TLB)
 - ➔ Hardwareunterstützung für verschachtelte Seitentabellen

Anmerkungen zu Folie 463:

- ➔ Zu den Schattentabellen: Der Hypervisor erstellt dabei für jede Seitentabelle eines Gast-BSs eine Tabelle, die die Seiten in die „richtigen“ Kacheln abbildet. Z.B. steht in der Seitentabelle eines Gast-BSs die Zuordnung „Seite 5 → Kachel 9“. Wenn Kachel 9 aber schon durch ein anderes Gast-BS belegt ist, ersetzt der Hypervisor diese durch eine andere freie Kachel, z.B. 12. In der Schattentabelle steht dann „Seite 5 → Kachel 12“. Wenn das Gast-BS seine Seitentabelle in die MMU laden will, führt der privilegierte Befehl zum Aufruf des Hypervisors, der stattdessen die zugehörige Schattentabelle in die MMU lädt.
- ➔ Bei der zweiten Lösung („Hinzufügen neuer Kacheln erst bei Seitenfehler“) kann das Gast-BS beliebige Seiten zu seiner Seitentabelle hinzufügen, ohne dass der Hypervisor eingreifen muß. Da sich dabei die Schattentabellen (die tatsächlich von der MMU genutzt werden) nicht ändern, gibt es beim Zugriff auf eine dieser Seiten einen Seitenfehler, den der Hypervisor dann behandeln kann. Beim Löschen einer Seite muß das Gast-BS immer einen privilegierten Befehl ausführen, der die Seite auch im TLB löscht.



Speichervirtualisierung ...

- ➔ Häufig: *Overcommitment*
 - ➔ alle VMs zusammen haben mehr Speicher als der Host
- ➔ Problem: Auslagerung durch Hypervisor nicht effizient / sinnvoll
 - ➔ unklar, welche Seiten für Gast-BS wichtig sind
 - ➔ Bei Auslagerung durch Gast-BS muss Hypervisor Seite ggf. erst wieder einlagern
- ➔ Lösung: *Balloon-Treiber*
 - ➔ im Gast-BS, kann ggf. fixierte Seiten allokiieren
 - ➔ (fixierte Seiten können nicht ausgelagert werden)
 - ➔ Gast-BS muss dafür „normale“ Seiten auslagern
- ➔ Z.T. auch Reduktion des Speicherverbrauchs durch Deduplikation



E/A-Virtualisierung

- ➔ Gast-BS bekommt virtuelle (Standard-)Geräte
 - ➔ z.B. virtuelle SATA-Platte, realisiert als normale Datei
 - ➔ für Netzwerk realisiert Hypervisor ggf. auch einen virtuellen Switch
- ➔ Heute auch Geräte mit Hardware-Unterstützung für Virtualisierung
 - ➔ Single-Root-I/O-Virtualisierung (SR-IOV)
 - ➔ Gerätecontroller bieten jeder VM eine eigene Schnittstelle (virtuelle Funktionen)
 - ➔ keine Mitwirkung des Hypervisors nötig

