

# Betriebssysteme (BS)

## VL 9 – Architekturen

**Daniel Lohmann**

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität  
Erlangen Nürnberg

WS 13 – 18. Dezember 2013

[http://www4.cs.fau.de/Lehre/WS13/V\\_BS](http://www4.cs.fau.de/Lehre/WS13/V_BS)

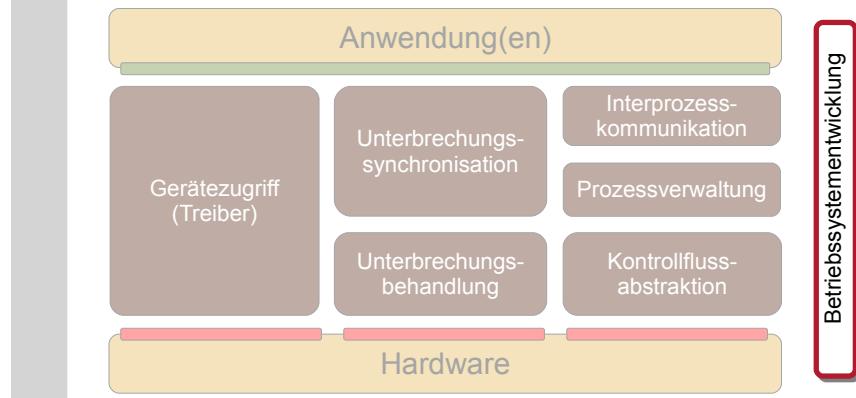


### Agenda

Einführung  
Geschichte, Mode und Trend  
Aktuelle Entwicklungen  
Zusammenfassung  
Referenzen



### Überblick: Einordnung dieser VL



### Agenda

Einführung  
Bewertungskriterien für Betriebssysteme  
Paradigmen der Betriebssystementwicklung  
Geschichte, Mode und Trend  
Aktuelle Entwicklungen  
Zusammenfassung  
Referenzen



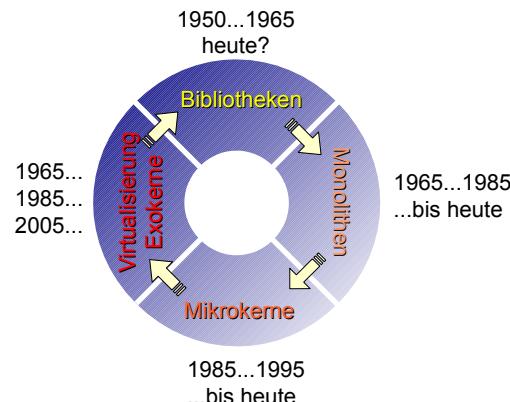
## Bewertungskriterien für Betriebssysteme

- Anwendungsorientierte Kriterien
  - **Portabilität**
    - Wie unabhängig ist man von der Hardware?
  - **Erweiterbarkeit**
    - Wie leicht lässt sich das System erweitern (z. B. um neue Gerätetreiber)?
  - **Robustheit**
    - Wie stark wirken sich Fehler in Einzelteilen auf das Gesamtsystem aus?
  - **Leistung**
    - Wie gut ist die Hardware durch die Anwendung auslastbar?
- Technische Kriterien (Architektur-eigenschaften)
  - **Isolationsmechanismus**
    - Wie werden Anwendungen / BS-Komponenten isoliert?
  - **Interaktionsmechanismus**
    - Wie kommunizieren Anwendungen / BS-Komponenten miteinander?
  - **Unterbrechungsmechanismus**
    - Wie werden Unterbrechungen zugestellt und bearbeitet?



## Betriebssystemgeschichte

### Paradigmen der Betriebssystementwicklung



## Betriebssystemgeschichte

### Paradigmen der Betriebssystementwicklung

#### Definition: Paradigma

Das Wort **Paradigma** [...] bedeutet „Beispiel“, „Vorbild“, „Muster“ oder „Abgrenzung“, „Vorurteil“, in allgemeinerer Form auch „Weitsicht“ oder „Weltanschauung“. [Wikipedia]



## Agenda

### Einführung

#### Geschichte, Mode und Trend

##### Bibliotheks-Betriebssysteme

Monolithen

Mikrokerne

Exokerne und Virtualisierung

##### Aktuelle Entwicklungen

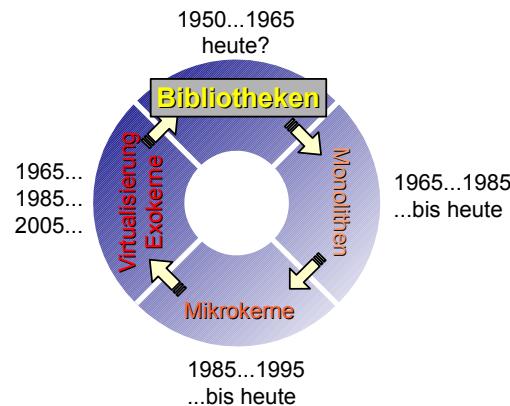
##### Zusammenfassung

##### Referenzen



## Überblick: Paradigmen

### Funktionsbibliotheken als einfache Infrastrukturen



## Entstehung von Bibliotheks-Betriebssystemen

### Logische Folge: Bibliotheks-Betriebssysteme

- Zusammenfassung von häufig benutzen Funktionen zur Ansteuerung von Geräten in **Software-Bibliotheken** (*Libraries*)
  - Systemfunktionen als „normale“ Subroutinen
- Funktionen der Bibliothek waren dokumentiert und getestet
  - verringerte Entwicklungszeit (von Anwendungen)
  - verringerte Übersetzungszeit (von Anwendungen)
- Bibliotheken konnten resident im Speicher des Rechners bleiben
  - verringter Speicherbedarf (der Anwendungen)
  - verringerte Ladezeit (der Anwendungen)
- Fehler konnten von Experten zentral behoben werden
  - verbesserte Zuverlässigkeit

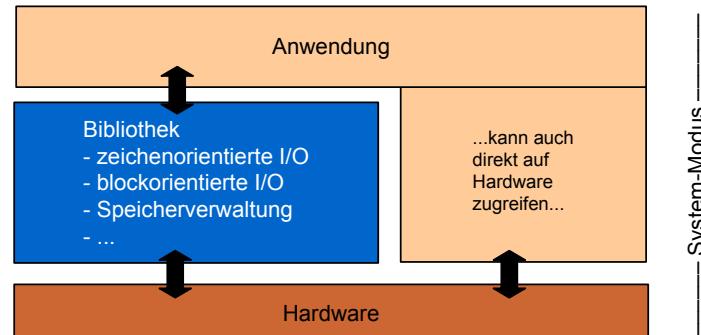


## Entstehung von Bibliotheks-Betriebssystemen

- Erste Rechnersysteme besaßen keinerlei Systemsoftware
  - Jedes Programm musste die gesamte Hardware selbst ansteuern
  - Systeme liefen Operator-gesteuert im Stapelbetrieb
    - single tasking, Lochkarten
  - Peripherie war vergleichsweise einfach
    - Seriell angesteuerter Lochkartenleser und -schreiber, Drucker, Bandlaufwerk
- Code zur Geräteansteuerung wurde in jedem Anwendungsprogramm repliziert
  - Die Folge war eine massive Verschwendungen von
    - Entwicklungszeit (teuer!)
    - Übersetzungszeit (sehr teuer!)
    - Speicherplatz (teuer!)
  - außerdem eine hohe Fehleranfälligkeit



## Architektur: Bibliotheks-Betriebssysteme



System-Modus



## Bewertung: Bibliotheks-Betriebssysteme

■ Anwendungsorientierte Kriterien	gering
■ <b>Portabilität</b> <ul style="list-style-type: none"><li>– keine Standards, eigene Bibliotheken für jede Architektur</li></ul>	mäßig
■ <b>Erweiterbarkeit</b> <ul style="list-style-type: none"><li>– theoretisch gut, in der Praxis oft „Spaghetti-Code“</li></ul>	sehr hoch
■ <b>Robustheit</b> <ul style="list-style-type: none"><li>– <i>single tasking</i>, Kosten für Programmwechsel sehr hoch</li></ul>	sehr hoch
■ <b>Leistung</b> <ul style="list-style-type: none"><li>– direktes Operieren auf der Hardware, keine Privilegien</li></ul>	
■ Technische Kriterien (Architektureigenschaften)	
■ <b>Isolationsmechanismus</b> <ul style="list-style-type: none"><li>– Anwendung <math>\equiv</math> System</li></ul>	nicht erforderlich
■ <b>Interaktionsmechanismus</b> <ul style="list-style-type: none"><li>– Betriebssystem <math>\equiv</math> Bibliothek</li></ul>	Funktionsaufrufe
■ <b>Unterbrechungsmechanismus</b> <ul style="list-style-type: none"><li>– Kommunikation mit Geräten über <i>polling</i></li></ul>	oft nicht vorhanden



## Agenda

### Einführung

#### Geschichte, Mode und Trend

Bibliotheks-Betriebssysteme

Monolithen

Mikrokerne

Exokerne und Virtualisierung

#### Aktuelle Entwicklungen

#### Zusammenfassung

Referenzen



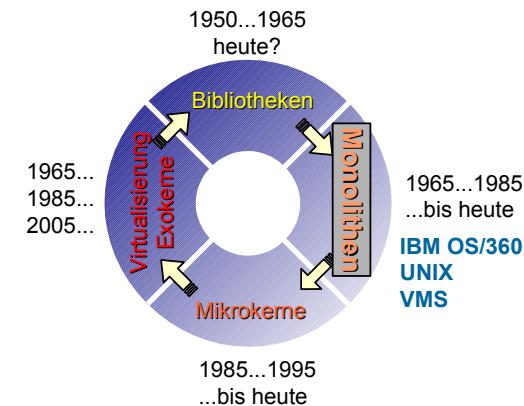
## Probleme: Bibliotheks-Betriebssysteme

■ Teure Hardware wird nicht optimal ausgelastet
■ Hoher Zeitaufwand beim Wechseln der Anwendung
■ Warten auf Ein-/Ausgabe verschwendet unnötig CPU-Zeit
■ Organisatorische Abläufe sehr langwierig
■ Stapelbetrieb, Warteschlangen
■ von der Abgabe eines Programms bis zum Erhalt der Ergebnisse vergehen oft Tage – um dann festzustellen, dass das Programm in der ersten Zeile einen Fehler hatte...
■ Keine Interaktivität möglich
■ Betrieb durch Operatoren, kein direkter Zugang zur Hardware
■ Programmabläufe nicht zur Laufzeit parametrierbar



## Überblick: Paradigmen

### Monolithen als Herrscher über das System

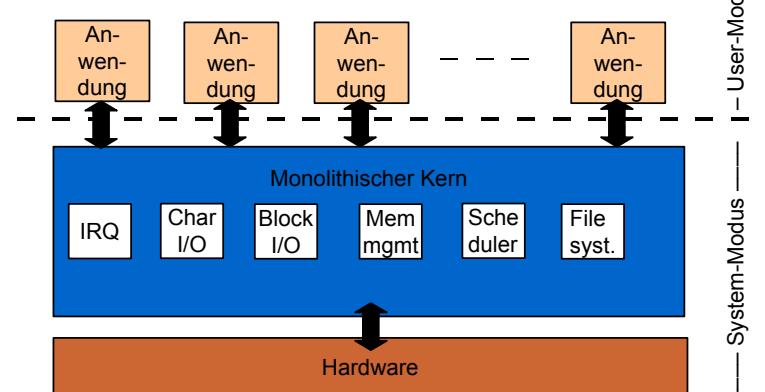


## Betriebssystem-Monolithen

- **Motivation:** Mehrprogrammbetrieb
- **Problem:** Isolation
- **Ansatz:** BS als Super-Programm, Kontrollinstanz
  - Programme laufen unter der Kontrolle des Betriebssystems
  - Dadurch erstmals (sinnvoll) Mehrprozess-Systeme realisierbar
- Einführung eines Privilegiensystems
  - Systemmodus ↔ Anwendungsmodus
  - Direkter Hardwarezugriff nur im Systemmodus
    - ↪ Gerätetreiber gehören zum System
- Einführung neuer Hard- und Software-Mechanismen
  - *Traps* in den Kern
  - Kontextumschaltung und -sicherung
  - *Scheduling* der Betriebsmittel



## Architektur: Monolithische Systeme



## Monolithische Systeme: IBM OS/360

- Eines der ersten monolithischen Betriebssysteme
  - Ziel: gemeinsames BS für alle IBM-Großrechner
  - Leistung und Speicher der Systeme differierten aber um Zehnerpotenzen zwischen „kleinen“ und „großen“ 360-Systemen
- Diverse Konfigurationen
  - PCP (Primary Control Program) 1965
    - Einprozessbetrieb, kleine Systeme
  - MFT (Multiprogramming with Fixed number of Tasks) 1966
    - mittlere Systeme (256 kB RAM)
    - feste Speicherpartitionierung zwischen Prozessen, feste Anzahl an Tasks
  - MVT (Multiprogramming with Variable number of Tasks): 1967
    - high end
    - Paging, optional Time Sharing Option (TSO) für interaktive Nutzung



## Monolithische Systeme: IBM OS/360



## Monolithische Systeme: IBM OS/360

- Richtungsweisende Konzepte
  - Hierarchisches Dateisystem
  - Prozesse können Unterprozesse erzeugen
  - Familienansatz: MFT und MVT sind von API und ABI her kompatibel
- Große Probleme bei der Entwicklung
  - Fred Brooks: „The Mythical Man-Month“ [6] [lesenswert!](#)
  - Problem der Konzeptuellen Integrität
    - Separation von Architektur und Implementierung war schwierig
  - „Second System Effect“
    - Entwickler wollten die „eierlegende Wollmilchsau“ bauen
  - Zu komplexe Abhängigkeiten zwischen Komponenten des Systems
    - Ab einer gewissen Codegröße blieb die Anzahl der Fehler konstant

~ Geburt der Softwaretechnik



## Monolithische Systeme: Bell Labs/AT&T UNIX



Historische PDP11



## Monolithische Systeme: Bell Labs/AT&T UNIX

- Ziel: Mehrprogrammbetrieb auf „kleinen“ Computern
  - Entwicklung seit Anfang der 70er Jahre
    - Kernelgröße im Jahr 1979 (7th Edition Unix, PDP11): ca. 50kB
  - von ursprünglich 2-3 Entwicklern geschrieben
    - überschaubar und handhabbar, ca. 10.000 Zeilen Quelltext
- Neu: Portabilität durch Hochsprache
  - C als domänenspezifische Sprache für Systemsoftware
  - UNIX wurde mit den Jahren auf nahezu jede Plattform portiert
- Weitere richtungsweisende Konzepte:
  - alles ist eine Datei, dargestellt als ein Strom von Bytes
  - komplexe Prozesse werden aus einfachen Programmen komponiert
    - Konzept der Pipe, Datenflussparadigma



## Monolithische Systeme: Bell Labs/AT&T UNIX

- Weitere Entwicklung von UNIX erfolgte stürmisch
  - Systeme mit großem Adressraum (VAX, RISC)
  - Der Kernel ist „mit gewachsen“ (System III, System V, BSD)
    - ohne wesentliche Strukturänderungen
  - Immer mehr komplexe Subsysteme wurden integriert
    - TCP/IP ist ungefähr so umfangreich wie der Rest des Kernels
- Linux orientiert(e) sich an der Struktur von System V
- UNIX war und ist einflussreich im akademischen Bereich durch frühe „Open Source“-Politik der Bell Labs
  - Viele Portierungen und Varianten entstanden
    - oftmals parallel zu Hardwareentwicklungen
  - In der akademischen Welt wurde UNIX zum Referenzsystem
    - Ausgleichspunkt und Vergleichssystem für alle neueren Ansätze



## Bewertung: Betriebssystem-Monolithen

- Anwendungsorientierte Kriterien
  - **Portabilität**
    - dank „C“ kann und konnte UNIX einfach portiert werden
  - **Erweiterbarkeit**
    - von Neukompilierung ↳ Modulkonzept
  - **Robustheit**
    - Anwendungen isoliert, nicht jedoch BS-Komponenten (Treiber!)
  - **Leistung**
    - Nur Betreten / Verlassen des Kerns ist teuer
- Technische Kriterien (Architektureigenschaften)
  - **Isolationsmechanismus** **Privilegierten, Addressräume**
    - Pro Anwendung ein Addressraum, Kern läuft auf Systemebene
  - **Interaktionsmechanismus** **Funktionsaufrufe, Traps**
    - Anwendung → Kern durch *Traps*, innerhalb des Kerns durch *call / ret*
  - **Unterbrechungsmechanismus** **Bearbeitung im Kern**
    - interne Unterteilung in UNIX: *bottom half, top half*



## Agenda

### Einführung

#### Geschichte, Mode und Trend

Bibliotheks-Betriebssysteme

Monolithen

Mikrokerne

Exokerne und Virtualisierung

#### Aktuelle Entwicklungen

#### Zusammenfassung

#### Referenzen



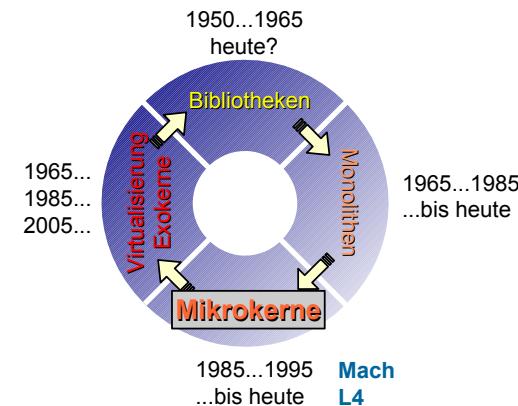
## Probleme: Betriebssystem-Monolithen

- Monolithen sind schwer handhabbar
  - Hinzufügen oder Ändern von Funktionalität betrifft oft mehr Module, als der Entwickler vorhergesehen hat
- Eingeschränkte Synchronisationsmechanismen
  - Oft nur ein „Big Kernel Lock“, d. h. nur ein Prozess kann zur selben Zeit im Kernmodus ausgeführt werden, alle anderen warten
  - Insbesondere bei Mehrprozessor-Systemen leistungsreduzierend
- Gemeinsamer Adressraum aller Kernkomponenten
  - Sicherheitsprobleme in einer Komponente (z.B. buffer overflow) führen zur Kompromittierung des gesamten Systems
  - Viele Komponenten laufen überflüssigerweise im Systemmodus
  - Komplexität und Anzahl von Treibern hat extrem zugenommen



## Überblick: Paradigmen

### Mikrokerne als Reduktion auf das Notwendige

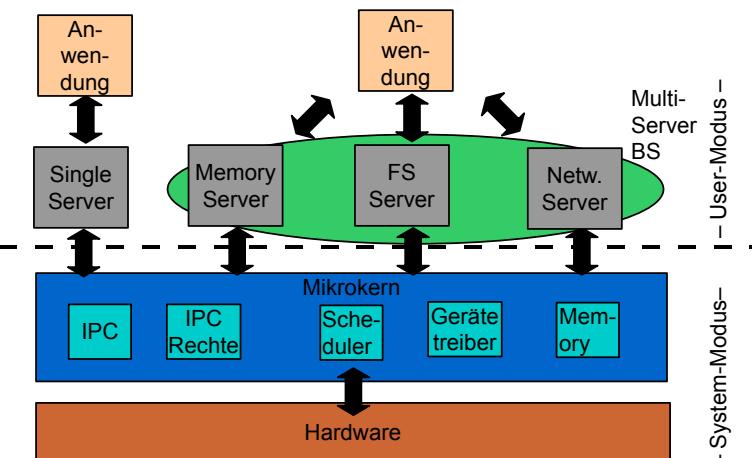


## Mikrokern-Betriebssysteme

- Ziel: Reduktion der Trusted Computing Base (TCB)
  - Minimierung der im privilegierten Modus ablaufenden Funktionalität
  - BS-Komponenten als Server-Prozesse im nichtprivilegierten Modus
  - Interaktion über Nachrichten (IPC, *Inter Process Communication*)
- Prinzip des geringsten Privilegs
  - Systemkomponenten müssen nur so viele Privilegien besitzen, wie zur Ausführung ihrer Aufgabe erforderlich sind
    - z.B. Treiber: Zugriff auf spezielle IO-Register, nicht auf die gesamte HW
  - Nur der Mikrokern läuft im Systemmodus
- Geringere Codegröße
  - L4: 10 kloc C++  $\leftrightarrow$  Linux: 1 Mloc C (ohne Treiber)
  - Ermöglicht Ansätze zur formalen Verifikation des Mikrokerns [10]



## Architektur: Mikrokerne erster Generation



## Mikrokerne erster Generation: CMU Mach [1]

- Ziel: Reduktion der TCB
- Ziel: Schaffung eines extrem portablen Systems
- Ziel: Verbesserung der Unix-Konzepte
  - Neue Kommunikationsmechanismen via IPC und Ports
    - Ports sind sichere IPC-Kommunikationskanäle
    - IPC ist optional netzwerktransparent: Unterstützung für verteilte Systeme
  - Parallele Aktivitäten innerhalb eines Prozessaddressraums
    - Unterstützung für Fäden  $\leadsto$  neuer Prozessbegriff als „Container“
    - Bessere Unterstützung für Mehrprozessorsysteme
  - Unterstützung „fremder“ Systemschnittstellen durch Personalities
- Ausgangspunkt: BSD UNIX
  - Schrittweise Separation der Funktionalität, die nicht im privilegierten Modus laufen muss in Benutzermodus-Prozesse
  - Anbindung über Ports und IPC



## Probleme: Mikrokerne erster Generation

- Probleme von Mach
  - hoher Overhead für IPC-Operationen
    - Systemaufrufe **Faktor 10** langsamer gegenüber monolithischem Kern
  - Immer noch viel zu große Code-Basis
    - Gerätetreiber und Rechteverwaltung für IPC im Mikrokern
    - $\leadsto$  die eigentlichen Probleme nicht gelöst!
  - Führte zu schlechtem Ruf von Mikrokernen allgemein
    - Einsetzbarkeit in der Praxis wurde bezweifelt
- Die Mikrokern-Idee galt Mitte der 90er Jahre als tot
  - Praktischer Einsatz von Mach erfolgte nur in hybriden Systemen
    - Separat entwickelte Komponenten für Mikrokern und Server
    - Kolokation der Komponenten in einem Adressraum, Ersetzen von in-kernel IPC durch Funktionsaufrufe
  - Bekanntestes Beispiel: Apple OS X  $\mapsto$  Mach 3 Mikrokern + FreeBSD



- **Ziel:** Mikrokern, diesmal aber richtig!
  - Verzicht auf Sekundärziele: Portabilität, Netzwerktransparenz, ...
- **Ansatz:** Reduktion auf das Notwendigste
  - Ein Konzept wird nur dann innerhalb des Mikrokerns toleriert, wenn seine Auslagerung die Implementierung verhindern würde.
  - synchroner IPC, Kontextwechsel, CPU Scheduler, Adressräume
- **Ansatz:** Gezielte Beschleunigung
  - fast IPCs durch Parameterübergabe in Registern
  - Gezielte Reduktion der Cache-Load (durch sehr kleinen Kern)
- Viele von Mikrokernen der 1. Generation noch im Systemmodus implementierte Funktionalität ausgelagert
  - z. B. Überprüfung von IPC-Kommunikationsrechten
  - vor allem aber: Treiber

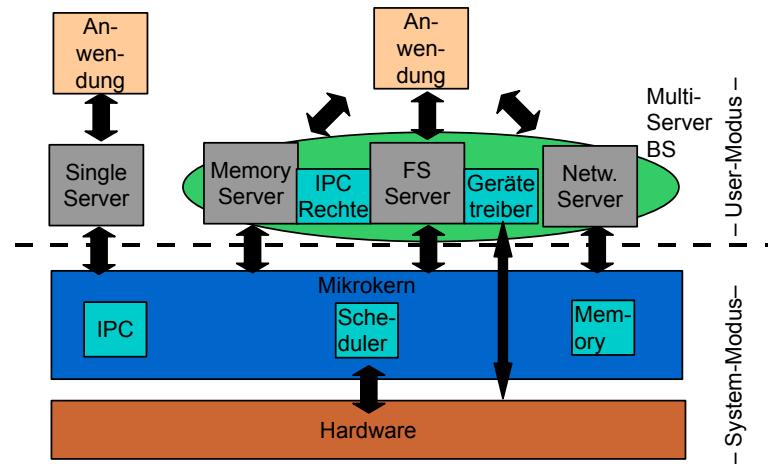


## Bewertung: Mikrokern-Betriebssysteme

- Anwendungsorientierte Kriterien
 

<ul style="list-style-type: none"> <li>■ <b>Portabilität</b> <ul style="list-style-type: none"> <li>– ursprünglich rein in Assembler, aktuell in C++ entwickelt</li> </ul> </li> <li>■ <b>Erweiterbarkeit</b> <ul style="list-style-type: none"> <li>– durch neue Server im Benutzermodus, auch zur Laufzeit</li> </ul> </li> <li>■ <b>Robustheit</b> <ul style="list-style-type: none"> <li>– durch strikte Isolierung</li> </ul> </li> <li>■ <b>Leistung</b> <ul style="list-style-type: none"> <li>– IPC-Performance ist <b>der</b> kritische Faktor</li> </ul> </li> </ul>	<b>mäßig</b>  <b>sehr hoch</b>  <b>sehr hoch</b>  <b>mäßig – gut</b>
--	--
- Technische Kriterien (Architektureigenschaften)
 

<ul style="list-style-type: none"> <li>■ <b>Isolationsmechanismus</b> <ul style="list-style-type: none"> <li>– Ein Addressraum pro Anwendung, ein Addressraum pro Systemkomponente</li> </ul> </li> <li>■ <b>Interaktionsmechanismus</b> <ul style="list-style-type: none"> <li>– Anwendungen und Systemkomponenten interagieren über Nachrichten</li> </ul> </li> <li>■ <b>Unterbrechungsmechanismus</b> <ul style="list-style-type: none"> <li>– Unterbrechungsbehandlung erfolgt durch Fäden im Benutzermodus</li> </ul> </li> </ul>	<b>Addressräume</b>  <b>IPC</b>  <b>IPC an Server-Prozess</b>
---	---



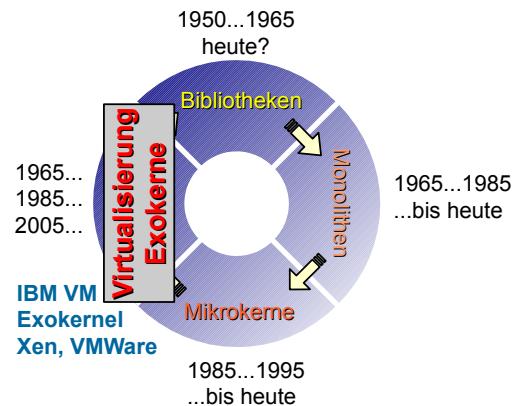
## Agenda

Einführung  
 Geschichte, Mode und Trend  
 Bibliotheks-Betriebssysteme  
 Monolithen  
 Mikrokerne  
 Exokerne und Virtualisierung  
 Aktuelle Entwicklungen  
 Zusammenfassung  
 Referenzen

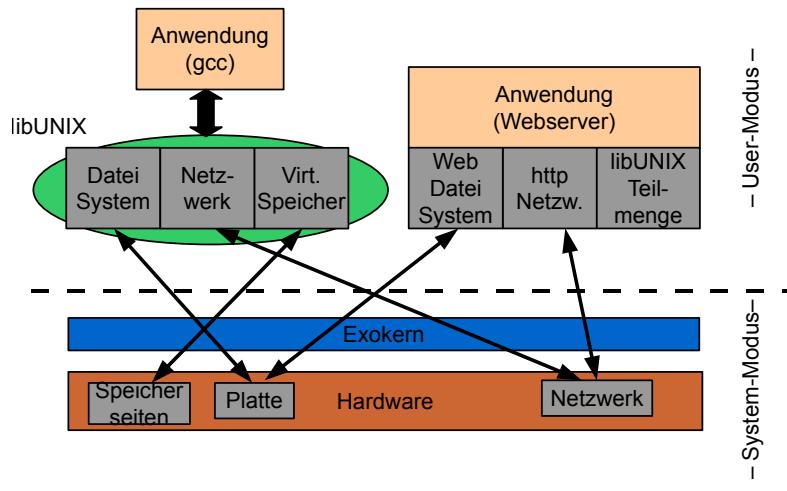


## Überblick: Paradigmen

### Exokerne und Virtualisierung als weitere Reduktion



## Architektur: Exokern-Betriebssysteme



## Exokern-Betriebssysteme: MIT exokernel

[8]

- Ziel: Leistungsverbesserung durch Reduktion
  - Entfernung von Abstraktionsebenen
  - Implementierung von Strategien (z.B. Scheduling) in der Anwendung
- Extrem kleiner Kern, dieser Implementiert nur
  - Schutz
  - Multiplexing von Ressourcen (CPU, Speicher, Disk-Blöcke, ...)
- Trennung von Schutz und Verwaltung der Ressourcen!
  - Keine Implementierung von IPC-Mechanismen (Mikrokerne) oder weiterer Abstraktionen (Monolithen)
  - Anwendungen können die *für sie* idealen Abstraktionen, Komponenten und Strategien verwenden



## Bewertung: Exokern-Betriebssysteme

- Anwendungsorientierte Kriterien
  - **Portabilität** sehr hoch
    - Exokerne sind sehr klein
  - **Erweiterbarkeit** sehr hoch
    - aber auch erforderlich! – der Exokern stellt kaum Funktionalität bereit
  - **Robustheit** gut
    - Schutz wird durch den Exokern bereitgestellt
  - **Leistung** sehr gut
    - Anwendungen operieren nahe an der Hardware, wenige Abstraktionsebenen
- Technische Kriterien (Architektureigenschaften)
  - **Isolationsmechanismus** Addressräume
    - Ein Addressraum pro Anwendung
      - + von ihr gebrauchter Systemkomponenten
  - **Interaktionsmechanismus** nicht vorgegeben
    - wird von der Anwendung bestimmt
  - **Unterbrechungsmechanismus** nicht vorgegeben
    - Exokern verhindert nur die Monopolisierung der CPU



## Probleme: Exokern-Betriebssysteme

- Exokernel sind nicht als Basis für die Verwendung mit beliebigen „legacy“-Anwendungen geeignet
- Anwendungen haben volle Kontrolle über Abstraktionen
  - müssen diese aber auch implementieren
  - hohe Anforderungen an Anwendungsentwickler
- Definition von Exokern-Schnittstellen ist schwierig
  - Bereitstellung adäquater Schnittstellen zur System-Hardware
  - Genaue Abwägung zwischen Mächtigkeit, Minimalismus und ausreichendem Schutz
- Bisher kein Einsatz in Produktionssystemen
  - Es existieren lediglich einige *proof-of-concept-Systeme*
  - Viele Fragen der Entwicklung von BS-Bibliotheken noch offen



## Virtualisierung: Beispiel IBM VM/370 (1972)

- Für IBM 360-Großrechner existierten mehrere Betriebssysteme
  - DOS/360, MVS: Stapel-orientierte Bibliotheks-Betriebssysteme
  - OS/360: Midrange Server-System
  - TSS/360: Interaktives Mehrbenutzersystem mit Time-Sharing
  - Kundenspezifische Entwicklungen
- Problem: wie kann man Anwendungen für all diese Systeme *gleichzeitig* verwenden?
  - Hardware war teuer (Millionen von US\$)
- Entwicklung der ersten Systemvirtualisierung „VM“ durch Kombination aus Emulation und Hardware-Unterstützung
  - Harte Partitionierung der Betriebsmittel
  - Gleichzeitiger Betrieb von stapelverarbeitenden und interaktiven Betriebssystemen wurde ermöglicht



## Virtualisierung

- Ziel: Isolation und Multiplexing **unterhalb** der Systemebene
- Ansatz: Virtual Machine Monitor (VMM) / Hypervisor
  - Softwarekomponente, läuft direkt auf der Hardware
  - stellt Abstraktion Virtual Maschine (VM) zur Verfügung
- VM simuliert die gesamten Hardware-Ressourcen
  - Prozessoren, Speicher, Festplatten, Netzwerkkarten, ...
  - Container für beliebige Betriebssysteme nebst Anwendungen
- Vergleich zu Exokernen
  - gröbere Granularität der zugeteilten Ressourcen
    - z.B. gesamte Festplattenpartition vs. einzelne Blöcke
  - „brute force“ Ansatz
    - Multiplexen ganzer Rechner statt einzelner Betriebsmittel
  - Anwendungen (und BS) brauchen nicht angepasst werden



## Virtualisierung von PCs: Beispiele VMWare, Xen (2003) [2]

- Ausgangslage: Problematik wie bei IBM in den 60er Jahren
  - Hardware wird immer leistungsfähiger – wohin mit den Ressourcen?
  - Ablauf mehrerer Betriebssystem-Instanzen gleichzeitig
  - Serverkonsolidierung, Kompatibilität zu Anwendungen
- Problem: IA-32 ist eigentlich nicht virtualisierbar
  - Virtualisierungskriterien von Popek und Goldberg [14] sind nicht erfüllt
  - Insbesondere: Äquivalenzanforderung – nicht alle Ring 0 Befehle trappen bei Ausführung auf Ring 3
- Ansatz: Paravirtualisierung
  - „kritische Befehle“ werden ersetzt
    - entweder zur Übersetzungszeit (Xen) oder zur Laufzeit (VMWare)
  - VMs laufen in Ring 3, Ringmodell durch Addressräume nachgebildet
    - Die meisten BS verwenden eh nur Ring 0 und Ring 3
- Neue IA-32 CPUs unterstützen Virtualisierung in HW (≈ VL 6)
  - Paravirtualisierung ist jedoch in der Praxis performanter



## Bewertung: Virtualisierung

- Anwendungsorientierte Kriterien
  - **Portabilität**
    - sehr hardware-spezifisch, Paravirtualisierung ist aufwändig
  - **Erweiterbarkeit**
    - in den üblichen VMs nicht vorgesehen
  - **Robustheit**
    - grobgranular auf der Ebene von VMs
  - **Leistung**
    - stark abhängig vom Einsatzszenario (CPU-lastig, IO-lastig, ...)
- Technische Kriterien (Architektur-eigenschaften)
  - **Isolationsmechanismus** **VM, Paravirtualisierung**
    - Jede Instanz bekommt einen eigenen Satz an Hardwaregeräten
  - **Interaktionsmechanismus** **nicht vorgesehen**
    - Anwendungen in den VMS kommunizieren miteinander über TCP/IP
  - **Unterbrechungsmechanismus** **Weiterleitung an VM**
    - VMM simuliert Unterbrechungen in den einzelnen VMs



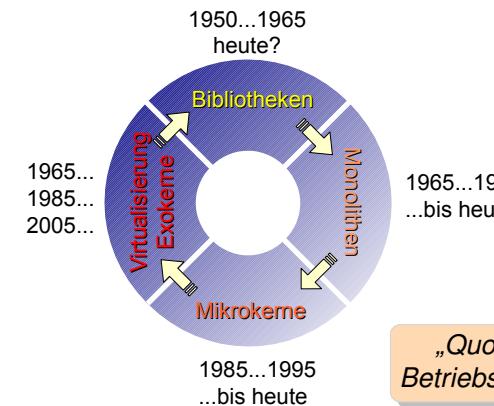
## Agenda

- Einführung
- Geschichte, Mode und Trend
- Aktuelle Entwicklungen
- Zusammenfassung
- Referenzen



## Überblick: Paradigmen

Back where we started?

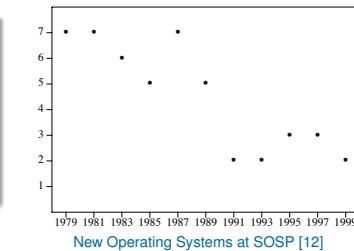


## Betriebssysteme ↪ ausgeforscht?

„Systems Software Research is Irrelevant“ [12]

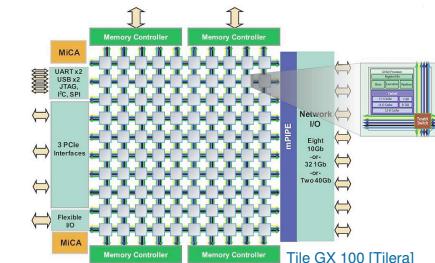
Urgestein Robert Pike (2000), einer der Entwickler von UNIX, Inferno [7], Plan 9 [13] und UTF-8 (zur Zeit bei Google beschäftigt):

- Where is the innovation? ↗ Microsoft, mostly
- Every other „new“ OS ends up being UNIX
- Linux? ↗ Just another copy of the same old stuff
- ...



Aber dann...

The Multicore  
Challenge!

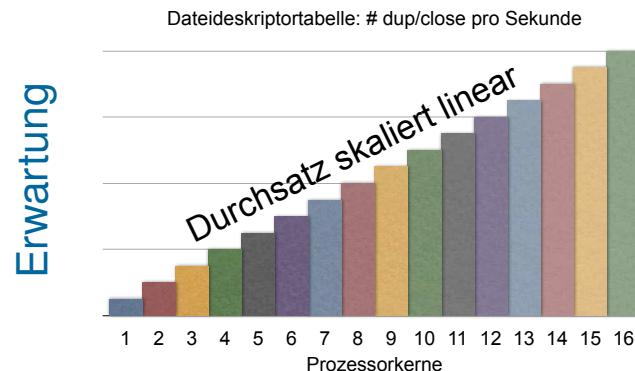


## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer et al. (OSDI 2008) [4]

- Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
- Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

```
int f = open(...); while(1){ close( dup( f ) ); }
```



## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer et al. (OSDI 2008) [4]

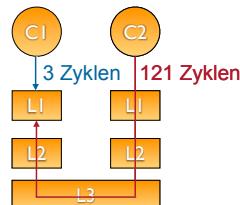
- Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
- Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

```
int f = open(...); while(1){ close( dup( f ) ); }
```

- Ergebnis: Schon ab **2 Kernen** **sinkt** der Gesamtdurchsatz
  - Grobgranulares *Locking*  $\rightsquigarrow$  *false sharing*  $\rightsquigarrow$  keine Skalierbarkeit
  - Geteilte Datenstruktur  $\rightsquigarrow$  *cache trashing*  $\rightsquigarrow$  Durchsatzabfall

```
fd_alloc () {
    lock(fd_table);
    fd = get_free_fd();
    set_fd_used(fd);
    fix_smallest_fd();
    unlock(fd_table);
}
```

1. *false sharing*



2. *cache trashing*



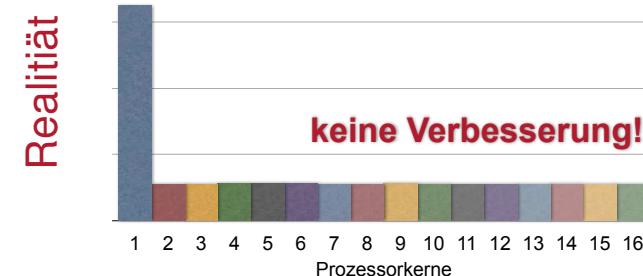
## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer et al. (OSDI 2008) [4]

- Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
- Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

```
int f = open(...); while(1){ close( dup( f ) ); }
```

Dateideskriptortabelle: # dup/close pro Sekunde



## Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer et al. (OSDI 2008) [4]

- Linux 2.6.25 auf 16-Kern AMD Opteron, 1–16 Kerne in Gebrauch
- Pro Kern ein Faden, der Dateideskriptoren anfordert und freigibt:
 

```
int f = open(...); while(1){ close( dup( f ) ); }
```

- Ergebnis: Schon ab **2 Kernen** **sinkt** der Gesamtdurchsatz
  - Grobgranulares *Locking*  $\rightsquigarrow$  *false sharing*  $\rightsquigarrow$  keine Skalierbarkeit
  - Geteilte Datenstruktur  $\rightsquigarrow$  *cache trashing*  $\rightsquigarrow$  Durchsatzabfall

**Multicore: POSIX (→ UNIX) considered harmful!**

„This problem is not specific to Linux, but is due to **POSIX semantics**, which require that a new file descriptor be visible to all of a process's threads even if only one thread uses it.“ [4]



## Folgerung: Wir brauchen neue Entwurfsansätze!

- **Corey** MIT, OSDI 2008, Exokern-artig: [4]
  - *Sharing* unter die Kontrolle der Applikation stellen
  - Datenstrukturen (im Normalfall) nur von einem Kern aus bearbeiten
  - Anwendungen müssen angepasst werden
- **Barrelyfish** ETH/MSR, SOSP 2009, Mikrokern-artig: [3]
  - BS als verteiltes System von Kernen verstehen und organisieren
  - kein implizites *Sharing*, Kommunikation nur über Nachrichten
- **Factored OS (fos)** MIT, 2009, Mikrokern-artig: [18]
  - BS für 100 bis 1000 Kerne  $\rightsquigarrow$  *time sharing* wird zu *space sharing*
  - Letztlich ähnlicher Ansatz wie Barrelyfish
- **TxOS** UT, SOSP 2009, Monolith (Linux): [15]
  - Konkurrenz zulassen durch *transactional syscalls* (statt *Locks*)
  - Anwendungen müssen angepasst werden



## ... oder doch nicht?

### Ergebnis: Alles nicht so schlimm...

„We find that we can remove most kernel bottlenecks that the applications stress by modifying the applications or kernel slightly. [...] the results suggest that **traditional kernel designs may be compatible with achieving scalability** on multicore computers.“ [5]

### Fazit

## Es bleibt spannend!

Systementwurf für Skalierbarkeit  $\rightsquigarrow$  [CC] (ab WS 2014).



## ... oder doch nicht?

- Boyd-Wickizer et al. (OSDI 2010) [5]
  - “An Analysis of Linux Scalability to Many Cores”
  - Skalierbarkeit von Linux 2.6.35-rc5 auf 48-Kern AMD Opteron
- Ansatz: *run – analyze – fix*
  - *run*: sieben „systemintensiven“ Anwendungen
    - Exim, memcached, Apache, PostgreSQL, gmake, Psearchy, MapReduce
  - *analyze*: gezielte Identifizierung von Flaschenhälzen
    - im Linux-Kern selber (16)
    - im Entwurf der Anwendung
    - durch die ungeschickte Verwendung der Systemschnittstelle
  - *fix*: Verbesserung, überwiegend durch Standardtechniken der parallelen Programmierung ( $\rightsquigarrow$  [PFP])



## Agenda

Einführung  
Geschichte, Mode und Trend  
Aktuelle Entwicklungen  
Zusammenfassung  
Referenzen



## Zusammenfassung: Betriebssystem-Architektur

- Betriebssysteme sind ein unendliches Forschungsthema
  - „alte“ Technologien (wie Virtualisierung oder Bibliotheken) finden immer wieder neue Einsatzgebiete
  - Hardwaretechnologie treibt die weitere Entwicklung
- Revolutionäre Neuerungen sind schwer durchzusetzen
  - Kompatibilität ist ein hohes Gut
    - Auf Anwendungsebene durch *Personalities* erreichbar
    - Neue Systeme scheitern jedoch meistens an fehlenden Treibern
  - Virtualisierte Hardware als Kompatibilitätsebene
- Die „ideale“ Architektur ist letztlich eine Frage der Anwendung!
  - Sensornetze, tief eingebettete Systeme  
Desktoprechner, Server, ...
  - Architektur  $\mapsto$  nichtfunktionale Eigenschaft des Betriebssystems



## Referenzen (Forts.)

- [6] Fred Brooks. *The Mythical Man Month*. Addison-Wesley, 1975. ISBN: 0-201-00650-2.
- [7] Sean Dorward, Rob Pike, Dave Presotto, et al. “The Inferno Operating System”. In: *Bell Labs Technical Journal* 2.1 (1997). URL: <http://www.vitanuova.com/inferno/papers/bltj.html>.
- [8] Dawson R. Engler, M. Frans Kaashoek, and James O’Toole. “Exokernel: An Operating System Architecture for Application-Level Resource Management”. In: *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP ’95)*. ACM SIGOPS Operating Systems Review. ACM Press, 1995, pp. 251–266. DOI: 10.1145/224057.224076.
- [9] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, et al. “The Performance of  $\mu$ -Kernel-Based Systems”. In: *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP ’97)*. New York, NY, USA: ACM Press, Oct. 1997. DOI: 10.1145/269005.266660.
- [10] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, et al. “seL4: formal verification of an OS kernel”. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP ’09)*. (Big Sky, Montana, USA). New York, NY, USA: ACM Press, 2009, pp. 207–220. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629596.



## Referenzen

- [1] Mike Accetta, Robert Baron, David Golub, et al. “MACH: A New Kernel Foundation for UNIX Development”. In: *Proceedings of the USENIX Summer Conference*. USENIX Association, June 1986, pp. 93–113.
- [2] Paul Barham, Boris Dragovic, Keir Fraser, et al. “Xen and the Art of Virtualization”. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP ’03)*. Vol. 37, 5. ACM SIGOPS Operating Systems Review. New York, NY, USA: ACM Press, Oct. 2003, pp. 164–177. DOI: 10.1145/945445.945462.
- [3] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, et al. “The multikernel: a new OS architecture for scalable multicore systems”. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP ’09)*. (Big Sky, Montana, USA). New York, NY, USA: ACM Press, 2009, pp. 29–44. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629579.
- [4] Silas Boyd-Wickizer, Haibo Chen, Rong Chen, et al. “Corey: An Operating System for Many Cores”. In: *8th Symposium on Operating System Design and Implementation (OSDI ’08)*. (San Diego, CA, USA). Berkeley, CA, USA: USENIX Association, 2008, pp. 43–57.
- [5] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, et al. “An Analysis of Linux Scalability to Many Cores”. In: *9th Symposium on Operating System Design and Implementation (OSDI ’10)*. (Vancouver, Canada). Berkeley, CA, USA, 2010.



## Referenzen (Forts.)

- [11] Jochen Liedtke. “On  $\mu$ -Kernel Construction”. In: *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP ’95)*. ACM SIGOPS Operating Systems Review. ACM Press, Dec. 1995. DOI: 10.1145/224057.224075.
- [PFP] Michael Philippse and Norbert Oster. *Parallele und Funktionale Programmierung*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 2, 2009 (jährlich). URL: <http://www2.informatik.uni-erlangen.de/teaching/WS2009/PFP/index.html>.
- [12] Rob Pike. *Systems Software Research is Irrelevant*. Talk. CS Colloquium, Columbia University. URL: <http://herpolhode.com/rob/utah2000.pdf> (visited on 12/09/2010).
- [13] Rob Pike, Dave Presotto, Sean Dorward, et al. “Plan 9 from Bell Labs”. In: *Computing Systems* 8.3 (1995), pp. 221–254.
- [14] Gerald J. Popek and Robert P. Goldberg. “Formal Requirements for Virtualizable Third Generation Architectures”. In: *Communications of the ACM* 17.7 (1974), pp. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073.



- [15] Donald E. Porter, Owen S. Hofmann, Christopher J. Rossbach, et al. "Operating System Transactions". In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. (Big Sky, Montana, USA). New York, NY, USA: ACM Press, 2009, pp. 161–176. ISBN: 978-1-60558-752-3. DOI: [10.1145/1629575.1629591](https://doi.org/10.1145/1629575.1629591).
- [16] *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. (Big Sky, Montana, USA). New York, NY, USA: ACM Press, 2009. ISBN: 978-1-60558-752-3.
- [CC] Wolfgang Schröder-Preikschat. *Concurrent Systems*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4, 2014 (jährlich). URL: [http://www4.informatik.uni-erlangen.de/Lehre/WS14/V\\_CC](http://www4.informatik.uni-erlangen.de/Lehre/WS14/V_CC).
- [17] Jim Smith and Ravi Nair. *Virtual Machines. Versatile Platforms for Systems and Processes*. Elsevier, 2005. ISBN: 978-1558609105.
- [18] David Wentzlaff and Anant Agarwal. "Factored operating systems (fos): the case for a scalable operating system for multicores". In: *ACM SIGOPS Operating Systems Review* 43 (2 Apr. 2009), pp. 76–85. ISSN: 0163-5980. DOI: [10.1145/1531793.1531805](https://doi.org/10.1145/1531793.1531805).

