

Betriebssystembau (BSB)

VL 14 – Zusammenfassung und Ausblick

Alexander Krause

Lehrstuhl für Informatik 12 – Arbeitsgruppe Systemsoftware / IRB
Technische Universität Dortmund

<https://sys.cs.tu-dortmund.de/de/lehre/ws25/bsb>

WS 25 – 26. Januar 2026

Agenda

Ziele und Zielerreichung

Betriebssystemforschung

SS 2026 am Lehrstuhl - VSS

Prüfung (

Abschlussarbeiten

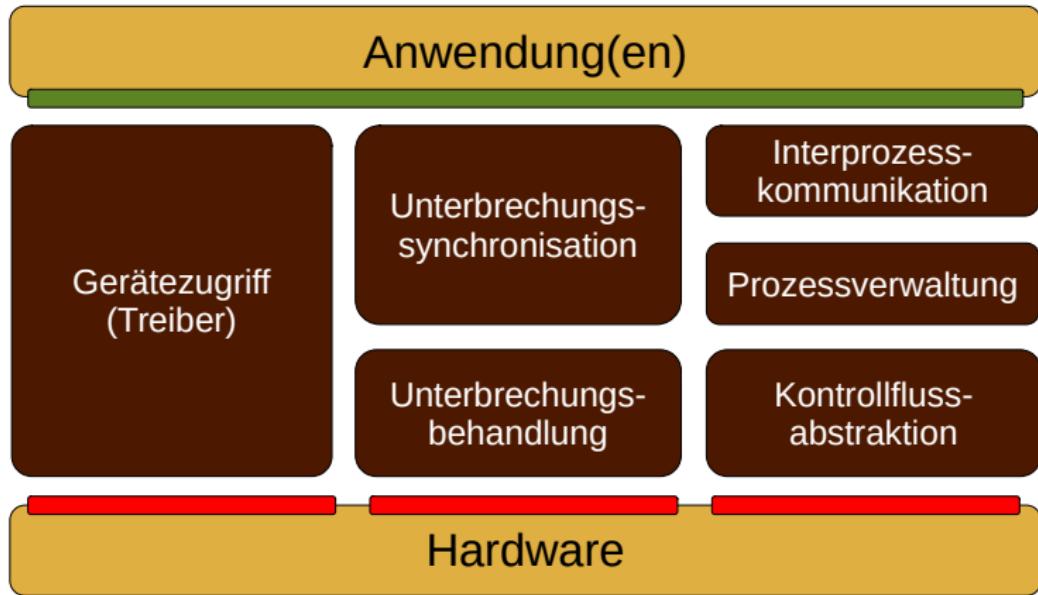
Ausblick

Referenzen



- **Vertiefen** des Wissens über die interne Funktionsweise von Betriebssystemen
 - Ausgangspunkt: Systemprogrammierung
 - Schwerpunkt: Nebenläufigkeit und Synchronisation
- **Entwickeln** eines Betriebssystems *von der Pike auf*
 - OOStuBS / MPStuBS Lehrbetriebssysteme
 - Praktische Erfahrungen im Betriebssystembau machen
- **Verstehen** der technologischen Hardware-Grundlagen
 - PC-Technologie verstehen und einschätzen können
 - Schwerpunkt: Intel x86_64





Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

VL₁ *Einführung*

VL₂ *BS-Entwicklung*

VL₃ *IRQs (Hardware)*

VL₄ *IRQs (Software)*

VL₅ *IRQs (Synchronisation)*

VL₆ *IRQs (SoftIRQ)*

VL₇ *Intel IA-32*

VL₈ *Koroutinen und Fäden*

VL₉ *Scheduling*

VL₁₀ *Architekturen*

VL₁₁ *Fadensynchronisation*

VL₁₂ *Gerätetreiber*

VL₁₃ *IPC*



1. Ein Streifzug durch die PC-Architektur

VL₁ *Einführung*

VL₂ *BS-Entwicklung*

VL₃ *IRQs (Hardware)*

VL₄ *IRQs (Software)*

VL₅ *IRQs (Synchronisation)*

VL₆ *IRQs (SoftIRQ)*

VL₇ *Intel IA-32*

VL₈ *Koroutinen und Fäden*

VL₉ *Scheduling*

VL₁₀ *Architekturen*

VL₁₁ *Fadensynchronisation*

VL₁₂ *Gerätetreiber*

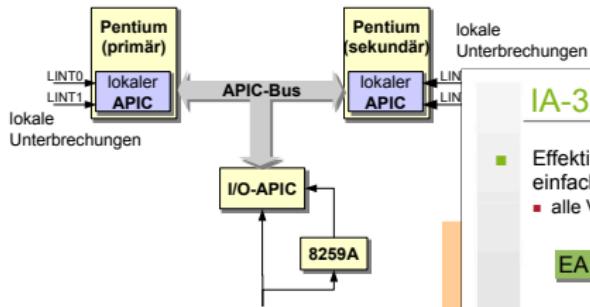
VL₁₃ *IPC*



1. Ein Streifzug durch die PC-Architektur

Die APIC Architektur

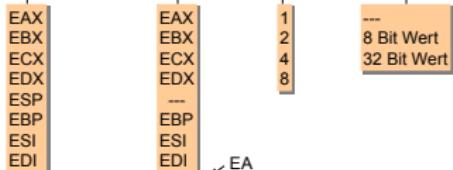
- ein APIC *Interrupt*-System besteht aus lokalen APICs auf jeder CPU und einem I/O APIC



IA-32: Adressierungsarten

- Effektive Adressen (EA) werden nach einem einfachen Schema gebildet
 - alle Vielzweckregister können dabei gleichwertig verwendet werden

$$EA = \text{Basis-Reg.} + (\text{Index-Reg.} * \text{Scale}) + \text{Displacement}$$



Beispiel: MOV EAX, Feld[ESI * 4]

- Lesen aus Feld mit 4 Byte großen Elementen und ESI als Index



2. Kontrollflüsse und ihre Interaktionen

VL₁ *Einführung*

VL₂ *BS-Entwicklung*

VL₃ *IRQs (Hardware)*

VL₄ *IRQs (Software)*

VL₅ *IRQs (Synchronisation)*

VL₆ *IRQs (SoftIRQ)*

VL₇ *Intel IA-32*

VL₈ *Koroutinen und Fäden*

VL₉ *Scheduling*

VL₁₀ *Architekturen*

VL₁₁ *Fadensynchronisation*

VL₁₂ *Gerätetreiber*

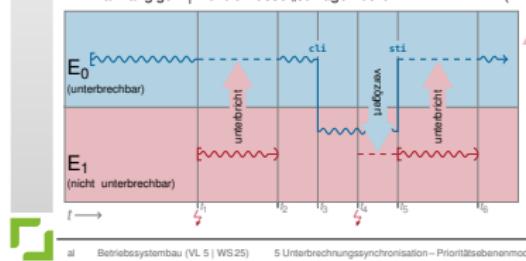
VL₁₃ *IPC*



2. Kontrollflüsse und ihre Interaktionen

Prioritätsebenenmodell

- Kontrollflüsse können die Ebene wechseln
 - Mit `c li` wechselt ein E_0 -Kontrollfluss explizit auf E_1
 - er ist ab dann nicht mehr unterbrechbar
 - andere E_1 -Kontrollflüsse werden verzögert
 - Mit `sti` wechselt ein E_1 -Kontrollfluss explizit auf E_0
 - er ist ab dann (wieder) unterbrechbar
 - anhängige E_0 -Kontrollflüsse „schlagen durch“



Erweitertes Prioritätsebenenmodell

Kontrollflüsse auf E_i werden

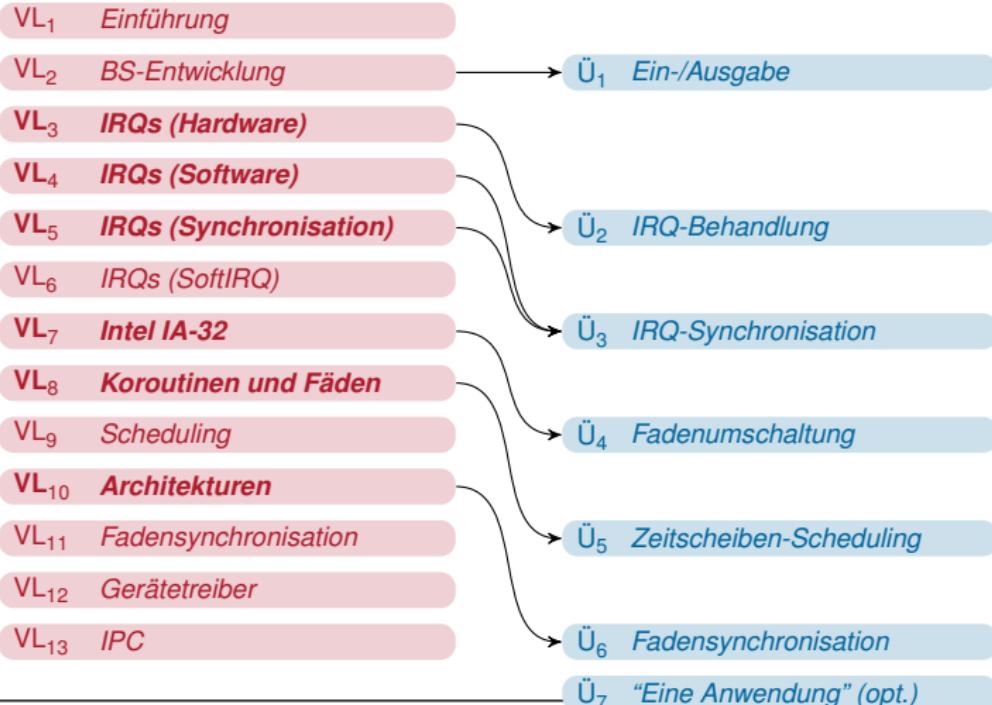
1. jederzeit unterbrochen durch Kontrollflüsse von E_m (für $m > i$)
2. nie unterbrochen durch Kontrollflüsse von E_k (für $k \leq i$)
3. jederzeit verdrängt durch Kontrollflüsse von E_l (für $l = 0$)



Kontrollflüsse der E_0 (Fadenebene) sind verdrängbar.

Für die Konsistenzsicherung auf dieser Ebene brauchen wir zusätzliche Mechanismen zur **Fadensynchronisation**.

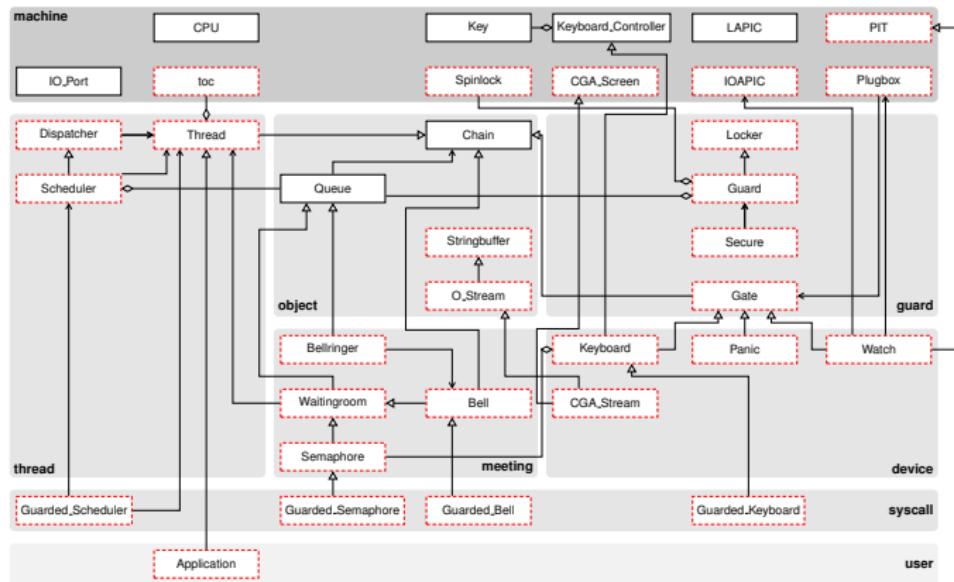
2. Kontrollflüsse und ihre Interaktionen



Was wir gemacht haben

Drei inhaltliche Schwerpunkte!

2. Kontrollflüsse und ihre Interaktionen



Generalization ("is a"):
Dependency ("uses a"):
Aggregation ("part of"):



al

3. BS-Konzept allgemein und am Beispiel (Windows/Linux)

VL₁ *Einführung*

VL₂ **BS-Entwicklung**

VL₃ *IRQs (Hardware)*

VL₄ *IRQs (Software)*

VL₅ *IRQs (Synchronisation)*

VL₆ *IRQs (SoftIRQ)*

VL₇ *Intel IA-32*

VL₈ **Koroutinen und Fäden**

VL₉ **Scheduling**

VL₁₀ **Architekturen**

VL₁₁ **Fadensynchronisation**

VL₁₂ **Gerätetreiber**

VL₁₃ **IPC**



3. BS-Konzept allgemein und am Beispiel (Windows/Linux)

Completely Fair Scheduler (CFS)

- **Ansatz:** Ablaufbereite Tasks bekommen die Rechenzeit gleichmäßig ("fair") zugeteilt
 - bei n Tasks jeweils $1/n$ -tel der CPU-Leistung
 - hierarchische Zuteilung durch *scheduling groups*
- CFS läuft nur bei **SCHED_NORMAL**
 - Echtzeittask (SCHED_RR und SCHED_FIFO) wie bisher
 - ansonsten: Task mit *geringster* CPU-Zeit hat höchste Priorität
- Planungskriterium ist die bislang zugeteilte CPU-Zeit
 - Ready-Liste als Rot-Schwarz-Baum, sortiert nach der Komplexität $O(\log N)$ (in der Praxis trotzdem effizienter als alter $O(1)$ -Scheiteln)
 - Prioritäten (im Sinne von nice) werden durch "schnellere/langsamere" Uhren abgebildet



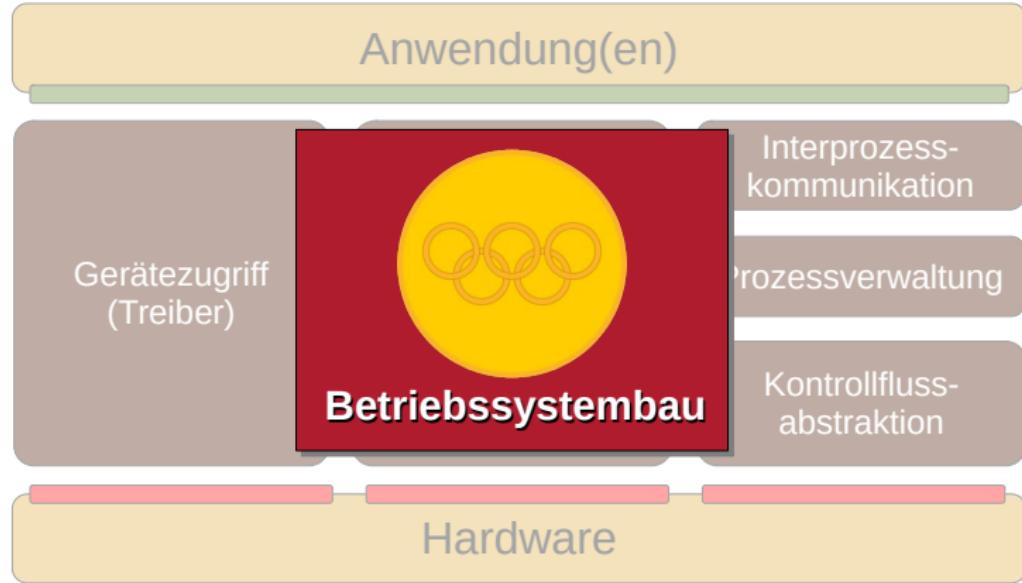
Windows – Treiberstruktur

Das E/A-System steuert den Treiber mit Hilfe der ...

- Initialisierungsroutine/Entladeroutine
 - wird nach/vor dem Laden/Entladen des Treibers ausgeführt
- Routine zum Hinzufügen von Geräten
 - PnP Manager hat ein neues Gerät für den Treiber
- "Verteilerroutinen"
 - Öffnen, Schließen, Lesen, Schreiben und gerätespezifische Oper.
- Interrupt Service Routine
 - wird von der zentralen Interrupt-Verteilungsroutine aufgerufen
- DPC-Routine
 - "Epilog" der Unterbrechungsbehandlung
- E/A-Komplettierungs- und -Abbruchroutine
 - Informationen über den Ausgang weitergeleiteter E/A-Aufträge



Zusammen eine ganze Menge!



Betriebssystementwicklung



Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- ...

↗ [BST]



Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- ...

↗ [BST]

Beispiel Linux [10]

Aug 91 Linux 0.01: bash, Dateisystem

Jan 92 Linux 0.12: Virtueller Speicher (Paging)



Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept ↗ [BST]
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- ...

Beispiel Linux [10]

Aug 91 Linux 0.01: bash, Dateisystem

Jan 92 Linux 0.12: Virtueller Speicher (Paging)

Mär 92 Linux 0.95: X-Windows, Unix Domain Sockets
(jetzt fehlte nur noch Netzwerk!)



Es fehlt noch eine ganze Menge!

- Adressraumverwaltung und Prozesskonzept → [BST]
- Dateisystem und Programmlader
- Netzwerk und TCP/IP
- ...

Beispiel Linux [10]

Aug 91 Linux 0.01: bash, Dateisystem

Jan 92 Linux 0.12: Virtueller Speicher (Paging)

Mär 92 Linux 0.95: X-Windows, Unix Domain Sockets
(jetzt fehlte nur noch Netzwerk!)

Mär 94 Linux 1.00: **Netzwerk und TCP/IP**



Agenda

Ziele und Zielerreichung

Betriebssystemforschung

SS 2026 am Lehrstuhl - VSS

Prüfung (

Abschlussarbeiten

Ausblick

Referenzen



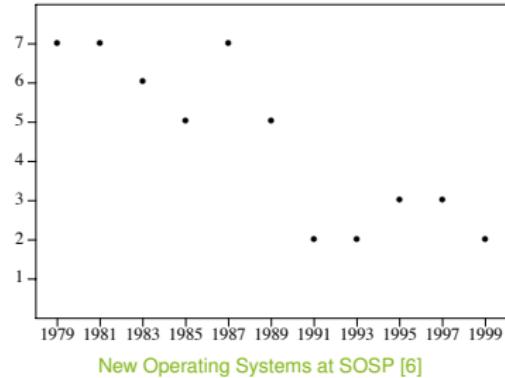
Betriebssysteme ↪ ausgeforscht?

„Systems Software Research is Irrelevant“

[6]

Urgestein Robert Pike (2000), einer der Entwickler von UNIX, Inferno [5], Plan 9 [7] 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
- ...



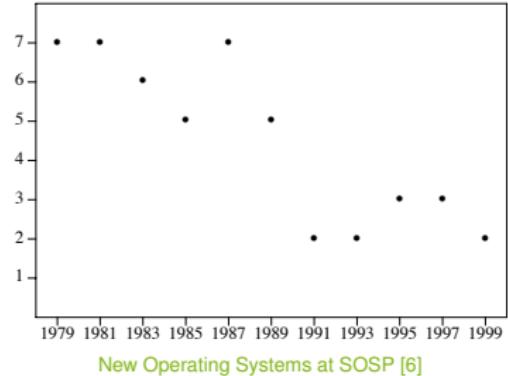
Betriebssysteme ↪ ausgeforscht?

„Systems Software Research is Irrelevant“

[6]

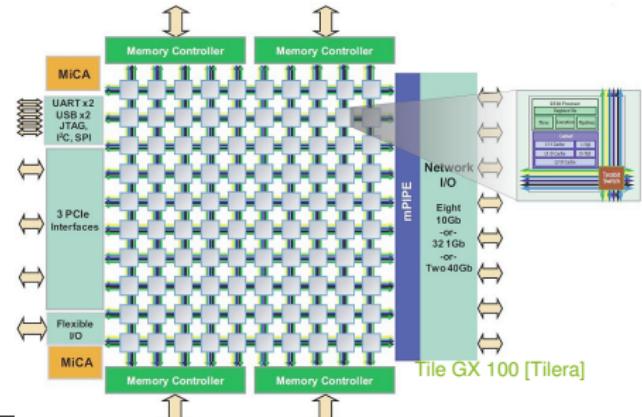
Urgestein Robert Pike (2000), einer der Entwickler von UNIX, Inferno [5], Plan 9 [7] und UTF-8
(zur Zeit bei Google beschäftigt):

- 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!*



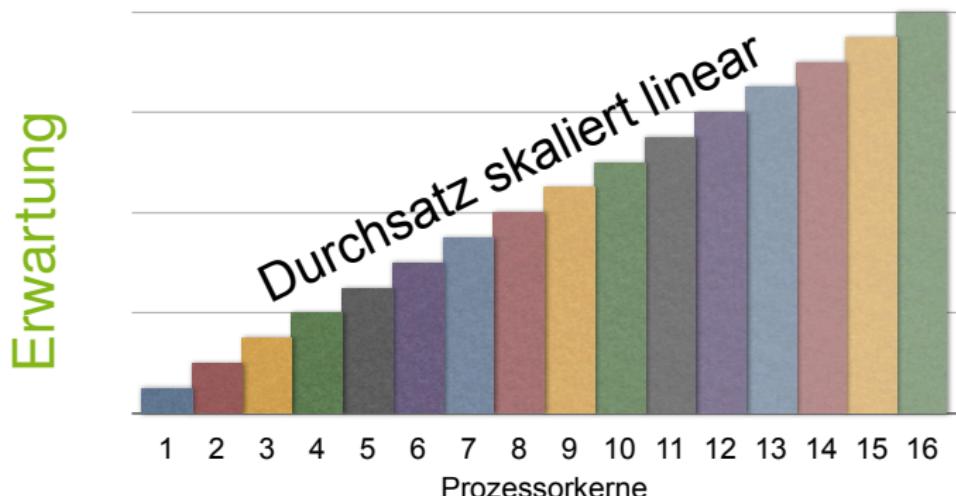
Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008)

[2]

- 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



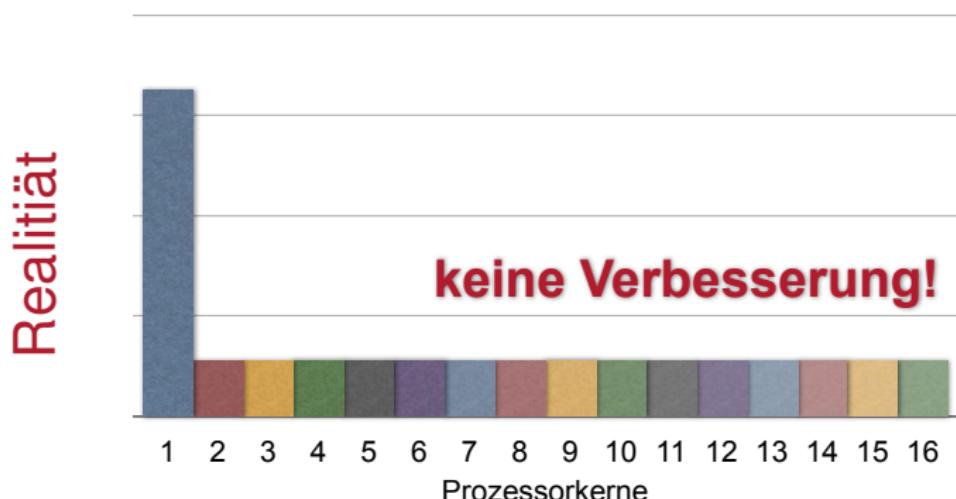
Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008)

[2]

- 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 u. a. (OSDI 2008) [2]
 - 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



Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008) [2]
 - 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
 1. Grobgranulares *Locking* ↗ *false sharing* ↗ keine Skalierbarkeit

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

1. *false sharing*



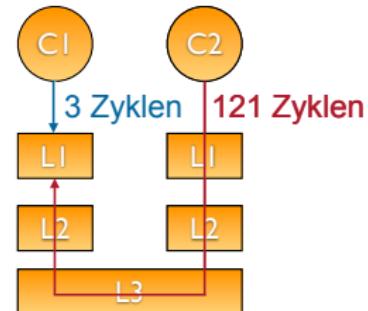
Fallstudie: Dateideskriptortabelle in Linux

- Boyd-Wickizer u. a. (OSDI 2008) [2]
 - 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
 1. Grobgranulares *Locking* ↗ *false sharing* ↗ keine Skalierbarkeit
 2. Geteilte Datenstruktur ↗ *cache trashing* ↗ 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 u. a. (OSDI 2008) [2]
 - 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
 - 1. Grobgranulares *Locking* ↗ *false sharing* ↗ keine Skalierbarkeit
 - 2. Geteilte Datenstruktur ↗ *cache trashing* ↗ 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.” [2]



Folgerung: Wir brauchen neue Entwurfsansätze!

- **Corey** MIT, OSDI 2008, Exokern-artig: [2]
 - *Sharing* unter die Kontrolle der Applikation stellen
 - Datenstrukturen (im Normalfall) nur von einem Kern aus bearbeiten
 - Anwendungen müssen angepasst werden
- **Barrelfish** ETH/MSR, SOSP 2009, Mikrokern-artig: [1]
 - BS als verteiltes System von Kernen verstehen und organisieren
 - kein implizites *Sharing*, Kommunikation nur über Nachrichten
- **Factored OS (fos)** MIT, 2009, Mikrokern-artig: [11]
 - BS für 100 bis 1000 Kerne \rightsquigarrow *time sharing* wird zu *space sharing*
 - Letztlich ähnlicher Ansatz wie Barrelfish
- **TxOS** UT, SOSP 2009, Monolith (Linux): [8]
 - Konkurrenz zulassen durch *transactional syscalls* (statt *Locks*)
 - Anwendungen müssen angepasst werden



- Boyd-Wickizer u. a. (OSDI 2010) [3]
 - „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 „systemintensive“ 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 (⇝ [PFP])



- Clements u. a. (SOSP 2013) [4]
 - „The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors“
 - Skalierbarkeit von *Schnittstellen* theoretisch und praktisch untersucht anhand Kommutativität der (möglichen) Implementierung.
- Idee: Wenn Operationen kommutativ sind, können sie (im Prinzip) auch skalierbar implementiert 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.” [3]

„Finally, using sv6, we showed that it **is practical to achieve a broadly scalable implementation of POSIX** by applying the rule, and that commutativity is essential to achieving scalability and performance on real hardware. ” [4]

Fazit

Es bleibt spannend!



Agenda

- Ziele und Zielerreichung
- Betriebssystemforschung
- SS 2026 am Lehrstuhl - VSS**
- Prüfung (✉)
- Abschlussarbeiten
- Ausblick
- Referenzen





Systemsoftware (von eingebetteten Systemen) **begeistert**

- Neue Verfahren und Architekturen zu entwickeln, ist spannend!
- Mikrokerne schotten Programme **räumlich** voneinander ab
- Verschlüsselungsalgorithmen garantieren **Datensicherheit**
- Echtzeitsysteme erlaubten ein **zeitlich** vorhersagbares Verhalten
 - Prioritätsorientierte Einplanung von Arbeitsaufträgen
- ...





Systemsoftware (von eingebetteten Systemen) **begeistert**

- Neue Verfahren und Architekturen zu entwickeln, ist spannend!
- Mikrokerne schotten Programme **räumlich** voneinander ab
- Verschlüsselungsalgorithmen garantieren **Datensicherheit**
- Echtzeitsysteme erlaubten ein **zeitlich** vorhersagbares Verhalten
 - Prioritätsorientierte Einplanung von Arbeitsaufträgen
- ...

Das ist jedoch nur die halbe Miete

- Erfordert möglichst fehlerfreie Implementierungen
- Implementierung muss mit Laufzeitfehlern umgehen können
 - Verfahren und Architekturen müssen **korrekt** arbeiten!



Wie lassen sich Fehler vermeiden bzw. behandeln?





Systemsoftware (

- Neue Verfahren u.
- Mikrokerne schotl
- Verschlüsselungs
- Echtzeitssysteme e
- Prioritätsorientie
- ...



Wie lassen sich Fehler vermeiden bzw. behan



Im Fokus dieser Veranstaltung: **Software**

1. Zuverlässige (robuste) Software entwickeln

- Robustheit gegenüber externen Fehlern (zur Laufzeit)
 - Wie erkenne und toleriere ich solche Fehler?
- Wie testet man, ob man korrekt mit solchen Fehlern umgeht?
- Hier „forschen“ wir (hoffentlich auch zusammen mit euch)



Im Fokus dieser Veranstaltung: **Software**

1. Zuverlässige (robuste) Software entwickeln

- Robustheit gegenüber externen Fehlern (zur Laufzeit)
 - Wie erkenne und toleriere ich solche Fehler?
- Wie testet man, ob man korrekt mit solchen Fehlern umgeht?
- Hier „forschen“ wir (hoffentlich auch zusammen mit euch)

2. Software zuverlässig entwickeln

- Wie kommt man zu einer möglichst fehlerfreien Implementierung?
- Welche Werkzeuge helfen mir dabei?
 - Was tun diese Werkzeuge eigentlich?
 - Welche Grenzen haben diese Werkzeuge demzufolge?
- Hier „lernen“ wir zusammen mit euch





Zuverlässige (robuste) Software entwickeln

- Maskieren von Fehlern durch Redundanz
 - Replizierte Ausführung
 - Homogene und heterogene Redundanz
- Härtung von Datenstrukturen und Kontrollfluss
 - Informationsredundanz
 - In Daten mithilfe von z.B. Prüfsummen
 - In Berechnungen/Kontrollfluss mithilfe arithmetischer Codierung
- Evaluierung von Fehlertoleranzmaßnahmen
 - Fehlerinjektion und Testen





Zuverlässige (robuste) Software entwickeln

- Maskieren von Fehlern durch Redundanz
 - Replizierte Ausführung
 - Homogene und heterogene Redundanz
- Härtung von Datenstrukturen und Kontrollfluss
 - Informationsredundanz
 - In Daten mithilfe von z.B. Prüfsummen
 - In Berechnungen/Kontrollfluss mithilfe arithmetischer Codierung
- Evaluierung von Fehlertoleranzmaßnahmen
 - Fehlerinjektion und Testen

Anknüpfungspunkte für den praktischen Einsatz aufzeigen

- Niemand braucht das 1001. Fehlertoleranzprotokoll!
 - Das den gegenwärtigen Stand der Kunst nicht reflektiert
 - Obendrein vielleicht fehlerhaft ist





Software zuverlässig entwickeln

- Typische Laufzeitfehler in C/C++-Programmen suchen und finden
 - Nullzeiger, Ganzzahlüberläufe, nicht initialisierte Speicherstellen, ...
→ Durch Testen oder mittels statischer Analysewerkzeuge
- Testüberdeckung: Wie gut hat man getestet?
 - die Testüberdeckung für ein gegebenes Programm messen
 - Gibt es Zusammenhänge zwischen der Testüberdeckung, der Testfallanzahl und anderen Metriken?
- Design-by-contract: statische, werkzeug-gestützte Verifikation
 - Formulierung/Verifikation von Nachbedingungen für kleine C-Programme
 - Mithilfe von Werkzeugen (AbsInt Astrée) wie sie auch Airbus einsetzt





Software zuverlässig entwickeln

- Typische **Laufzeitfehler** in C/C++-Programmen suchen und finden
 - Nullzeiger, Ganzzahlüberläufe, nicht initialisierte Speicherstellen, ...
→ Durch Testen oder mittels statischer Analysewerkzeuge
- **Testüberdeckung:** Wie gut hat man getestet?
 - die Testüberdeckung für ein gegebenes Programm messen
 - Gibt es Zusammenhänge zwischen der Testüberdeckung, der Testfallanzahl und anderen Metriken?
- **Design-by-contract:** statische, werkzeug-gestützte Verifikation
 - Formulierung/Verifikation von Nachbedingungen für kleine C-Programme
 - Mithilfe von Werkzeugen (AbsInt Astrée) wie sie auch Airbus einsetzt



Vorurteile gegenüber formalen Methoden abbauen

- Keine **unverwendbaren Monster** mehr
 - Vollbringen aber auch **keine Wunder**
 - Ihre Anwendung ist noch immer mühsam, aber sie lohnt sich



Agenda

Ziele und Zielerreichung
Betriebssystemforschung
SS 2026 am Lehrstuhl - VSS
Prüfung (💀)
Abschlussarbeiten
Ausblick
Referenzen



- 30 min. Fachgespräch über Betriebssystembau
- Termine
 - 23. + 24. Februar
 - 24. + 25. März
- Anmeldung
 - Anmeldung für mündl. Prüfung ausfüllen und ausdrucken
 - Termin und Unterschrift von mir abholen
 - Wir senden den Zettel an das PA





Geprüft wird der Stoff der Vorlesung

- ihr müsst **nicht** den Quellcode (auswendig) kennen
- aber das Prinzip müsst ihr erklären können!
- übt mit Kommilitonen, erklärt euch gegenseitig die Vorgehensweise



- Geprüft wird der Stoff der Vorlesung
 - ihr müsst **nicht** den Quellcode (auswendig) kennen
 - aber das Prinzip müsst ihr erklären können!
 - übt mit Kommilitonen, erklärt euch gegenseitig die Vorgehensweise
- alte Prüfungsprotokolle online bei der FSI Informatik:
<https://fsi.cs.fau.de/dw/pruefungen/hauptstudium/ls4/bs>



- Geprüft wird der Stoff der Vorlesung
 - ihr müsst **nicht** den Quellcode (auswendig) kennen
 - aber das Prinzip müsst ihr erklären können!
 - übt mit Kommilitonen, erklärt euch gegenseitig die Vorgehensweise
- alte Prüfungsprotokolle online bei der FSI Informatik:
<https://fsi.cs.fau.de/dw/pruefungen/hauptstudium/ls4/bs>
- *bei Prüfungsabsage:* Bitte immer eine (kurze) Mail
 - immer. Egal wie kurzfristig.
 - aber je früher desto besser
 - ggf. auch gleich Wunschersatztermin



- kommt [über]pünktlich
- und ausgeschlafen ☺
- ein Prüfer und ein protokollierender Beisitzer
- statt schweigend zu denken, lieber eure Überlegung aussprechen
⇒ man darf nur bepunktten, was ihr von euch gebt
(und der Prüfer kann euch auf den richtigen Weg bringen)
- sollten Worte fehlen/nicht ausreichen, so habt ihr Stift und Papier
- die 30 Minuten sind schnell vorbei
- ihr bekommt nach weiteren 1-10 Minuten eure Note



Agenda

Ziele und Zielerreichung
Betriebssystemforschung
SS 2026 am Lehrstuhl - VSS
Prüfung ()
Abschlussarbeiten
Ausblick
Referenzen



Zur Zeit im Angebot:

- (Bachelorarbeiten)
- Masterarbeiten

<https://sys.cs.tu-dortmund.de/de/lehre/abschlussarbeiten/>



Zur Zeit im Angebot:

- (Bachelorarbeiten)
- Masterarbeiten

<https://sys.cs.tu-dortmund.de/de/lehre/abschlussarbeiten/>
→ Ist ok, aber lieber ...



Zur Zeit im Angebot:

- (Bachelorarbeiten)
- Masterarbeiten

<https://sys.cs.tu-dortmund.de/de/lehre/abschlussarbeiten/>

→ Ist ok, aber lieber ...

... persönlich nachfragen...!



Das BSB-Team wünscht
erfolgreiche und erholsame
„Semesterferien“

... und ein Wiedersehen
im Sommersemester 2026!



Referenzen

- [1] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand u. a. „The multikernel: a new OS architecture for scalable multicore systems“. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. ACM Press. Big Sky, MT, USA: ACM Press, Okt. 2009, S. 29–44. isbn: 978-1-60558-752-3. doi: 10.1145/1629575.1629579.
- [2] Silas Boyd-Wickizer, Haibo Chen, Rong Chen u. a. „Corey: An Operating System for Many Cores“. In: *8th Symposium on Operating System Design and Implementation (OSDI '08)*. USENIX Association. San Diego, CA, USA: USENIX Association, Dez. 2008, S. 43–57. isbn: 978-1-931971-65-2. url: https://www.usenix.org/legacy/event/osdi08/tech/full_papers/boyd-wickizer/boyd_wickizer.pdf.
- [3] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao u. a. „An Analysis of Linux Scalability to Many Cores“. In: *9th Symposium on Operating System Design and Implementation (OSDI '10)*. USENIX Association. Vancouver, BC, Canada: USENIX Association, Okt. 2010. isbn: 978-1-931971-79-9.
- [4] Austin T. Clements, M. Frans Kaashoek, Nickolai Zeldovich u. a. „The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors“. In: *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)* (Farmington, PA, USA). ACM Press. New York, NY, USA: ACM Press, 2013, S. 1–17. isbn: 978-1-4503-2388-8. doi: 10.1145/2517349.2522712.



- [5] Sean Dorward, Rob Pike, Dave Presotto u. a. „The Inferno Operating System“. In: *Bell Labs Technical Journal* 2.1 (Winter 1997).
- [PFP] Norbert Oster. *Parallele und Funktionale Programmierung*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 2, 2015 (jährlich). url: <https://www2.cs.fau.de/teaching/SS2015/PFP/index.html>.
- [6] Rob Pike. *Systems Software Research is Irrelevant*. Talk. CS Colloquium, Columbia University. url: <http://herpolhode.com/rob/utah2000.pdf> (besucht am 09.12.2010).
- [7] Rob Pike, Dave Presotto, Sean Dorward u. a. „Plan 9 from Bell Labs“. In: *Computing Systems* 8.3 (1995), S. 221–254.
- [8] Donald E. Porter, Owen S. Hofmann, Christopher J. Rossbach u. a. „Operating System Transactions“. In: *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. ACM Press. Big Sky, MT, USA: ACM Press, Okt. 2009, S. 161–176. isbn: 978-1-60558-752-3. doi: 10.1145/1629575.1629591.
- [9] *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. ACM Press. Big Sky, MT, USA: ACM Press, Okt. 2009. isbn: 978-1-60558-752-3.
- [10] Linus Torvalds und David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. HarperCollins, 2001. isbn: 978-0066620725.



- [BST] Peter Ulbrich. *Betriebssystemtechnik*. Vorlesung mit Übung. Technische Universität Dortmund, Lehrstuhl für Informatik 12, 2024 (jährlich). url: <https://sys.cs.tu-dortmund.de/de/lehre/>.
- [11] David Wentzlaff und 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), S. 76–85. issn: 0163-5980. doi: 10.1145/1531793.1531805.

