

# Betriebssysteme (BS)

## VL 14 – Zusammenfassung und Ausblick

**Volkmar Sieh / Daniel Lohmann**

Lehrstuhl für Informatik 4  
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität  
Erlangen Nürnberg

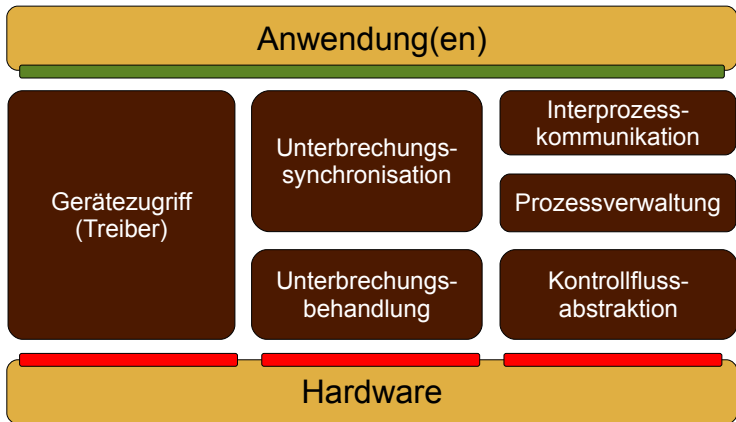
WS 21 – 7. Februar 2022



[https://www4.cs.fau.de/Lehre/WS21/V\\_BS](https://www4.cs.fau.de/Lehre/WS21/V_BS)

- **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





VL<sub>1</sub> **Einführung**

VL<sub>2</sub> **BS-Entwicklung**

VL<sub>3</sub> **IRQs (Hardware)**

VL<sub>4</sub> **IRQs (Software)**

VL<sub>5</sub> **IRQs (SoftIRQ)**

VL<sub>6</sub> **IRQs (Synchronisation)**

VL<sub>7</sub> **Intel IA-32**

VL<sub>8</sub> **Koroutinen und Fäden**

VL<sub>9</sub> **Scheduling**

VL<sub>10</sub> **Architekturen**

VL<sub>11</sub> **Fadensynchronisation**

VL<sub>12</sub> **Gerätetreiber**

VL<sub>13</sub> **IPC**



## 1. Ein Streifzug durch die PC-Architektur

VL<sub>1</sub> **Einführung**

VL<sub>2</sub> **BS-Entwicklung**

VL<sub>3</sub> **IRQs (Hardware)**

VL<sub>4</sub> **IRQs (Software)**

VL<sub>5</sub> **IRQs (SoftIRQ)**

VL<sub>6</sub> **IRQs (Synchronisation)**

VL<sub>7</sub> **Intel IA-32**

VL<sub>8</sub> **Koroutinen und Fäden**

VL<sub>9</sub> **Scheduling**

VL<sub>10</sub> **Architekturen**

VL<sub>11</sub> **Fadensynchronisation**

VL<sub>12</sub> **Gerätetreiber**

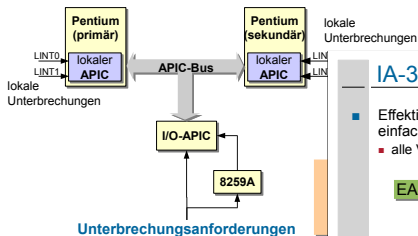
VL<sub>13</sub> **IPC**



### 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



vs/dl Betriebssysteme (VL 3 | WS 20) 3 Unterbrechungen, Hardware – Hardware-Ar

#### IA-32: Adressierungsarten

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

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

EAX  
EBX  
ECX  
EDX  
ESP  
EBP  
ESI  
EDI

1  
2  
4  
8

EA

8 Bit Wert  
32 Bit Wert

- Beispiel: `MOV EAX, Feld[ESI * 4]`
  - Lesen aus Feld mit 4 Byte großen Elementen und ESI als Index



vs/dl Betriebssysteme (VL 7 | WS 20) 7 IA-32 – Die 32-Bit Intel-Architektur

7-19



## 2. Kontrollflüsse und ihre Interaktionen

VL<sub>1</sub> **Einführung**

VL<sub>2</sub> **BS-Entwicklung**

VL<sub>3</sub> **IRQs (Hardware)**

VL<sub>4</sub> **IRQs (Software)**

VL<sub>5</sub> **IRQs (SoftIRQ)**

VL<sub>6</sub> **IRQs (Synchronisation)**

VL<sub>7</sub> **Intel IA-32**

VL<sub>8</sub> **Koroutinen und Fäden**

VL<sub>9</sub> **Scheduling**

VL<sub>10</sub> **Architekturen**

VL<sub>11</sub> **Fadensynchronisation**

VL<sub>12</sub> **Gerätetreiber**

VL<sub>13</sub> **IPC**

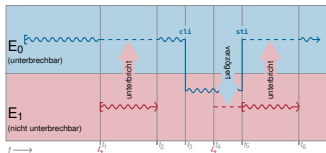


## 2. Kontrollflüsse und ihre Interaktionen

### Prioritätsebenenmodell

#### ■ Kontrollflüsse können die Ebene wechseln

- Mit  $cli$  **wechselt** ein  $E_0$ -Kontrollfluss explizit auf  $E_1$ 
  - er ist ab dann nicht mehr unterbrechbar
  - andere  $E_1$ -Kontrollflüsse werden verzögert (↔ Sequentialisierung)
- Mit  $sti$  **wechselt** ein  $E_1$ -Kontrollfluss explizit auf  $E_0$ 
  - er ist ab dann (wieder) unterbrechbar
  - anhängige  $E_1$ -Kontrollflüsse „schlagen durch“ (↔ S)



vs/dl Betriebssysteme (VL 6 | WS 20) 6 Unterbrechungen, Synchronisation – Priorität

### Erweitertes Prioritätsebenenmodell

#### ■ Kontrollflüsse auf $E_l$ werden

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



Kontrollflüsse der  $E_0$  (Faden-ebene) sind **verdrängbar**.

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



vs/dl Betriebssysteme (VL 11 | WS 20) 11 Fadensynchronisation – Prioritätsebenenmodell mit Fäden 11–10

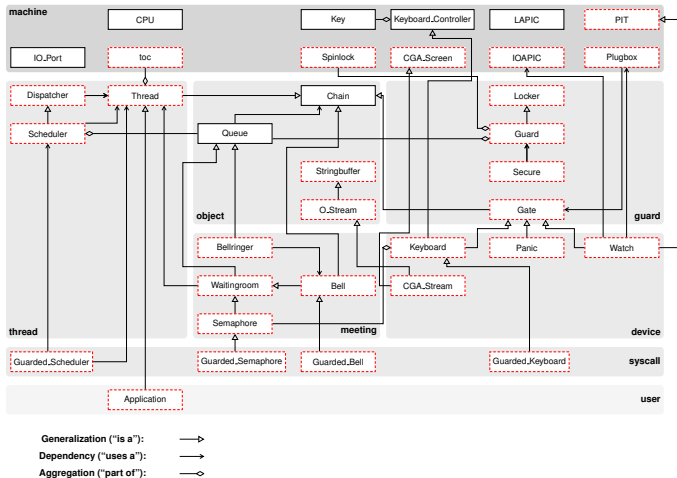




## 2. Kontrollflüsse und ihre Interaktionen



## 2. Kontrollflüsse und ihre Interaktionen



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

VL<sub>1</sub> *Einführung*

VL<sub>2</sub> *BS-Entwicklung*

VL<sub>3</sub> *IRQs (Hardware)*

VL<sub>4</sub> *IRQs (Software)*

VL<sub>5</sub> *IRQs (SoftIRQ)*

VL<sub>6</sub> *IRQs (Synchronisation)*

VL<sub>7</sub> *Intel IA-32*

VL<sub>8</sub> *Koroutinen und Fäden*

VL<sub>9</sub> *Scheduling*

VL<sub>10</sub> *Architekturen*

VL<sub>11</sub> *Fadensynchronisation*

VL<sub>12</sub> *Gerätetreiber*

VL<sub>13</sub> *IPC*



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

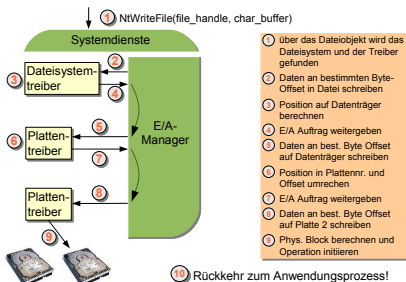
#### Completely Fair Scheduler (CFS)

- 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) w
  - ansonsten: Task mit *geringster* CPU-Zeit hat höc
- Scheduling-Kriterium ist die bislang zugeteilte
  - Ready-Liste als Rot-Schwarz-Baum, sortiert nac
  - Komplexität  $O(\log N)$   
(in der Praxis trotzdem effizienter als alter  $O(1)-S$ )
  - Prioritäten (im Sinne von nice) werden durch "schnellere/langsamere" Uhren abgebildet



vs/dl Betriebssysteme (VL 9 | WS 20) 9 Fadenverwaltung – Ablaufplanung

#### Windows – typischer E/A-Ablauf

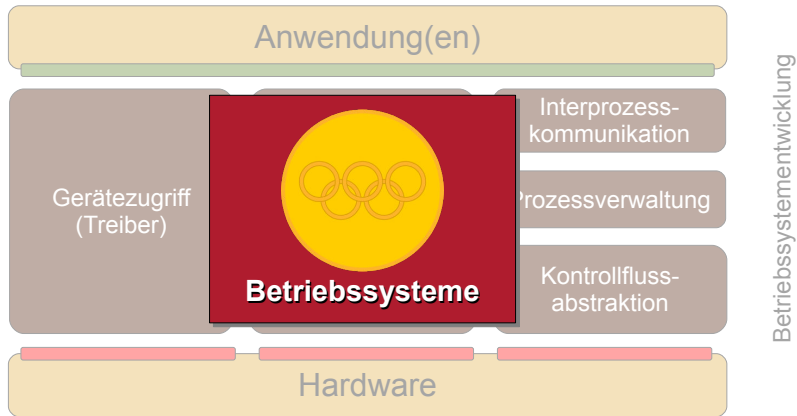


vs/dl Betriebssysteme (VL 12 | WS 20) 12 Treiber – Struktur des E/A-Systems

12-29



# Zusammen eine ganze Menge!



## Es fehlt noch eine ganze Menge!

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



## Es fehlt noch eine ganze Menge!

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

### Beispiel Linux [14]

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**



# Betriebssysteme $\mapsto$ ausgeforscht?

*„Systems Software Research is Irrelevant“* [9]

Urgestein Robert Pike (2000), einer der Entwickler von UNIX, Inferno [5], Plan 9 [10] und UTF-8 (zur Zeit bei Google beschäftigt):

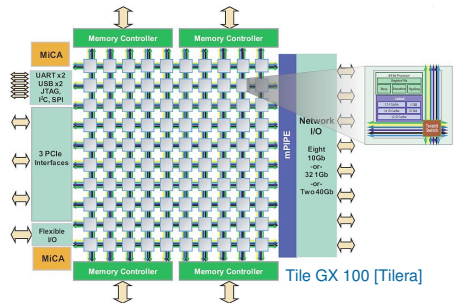
- Where is the innovation?  $\leadsto$  Microsoft, mostly
- Every other „new“ OS ends up being UNIX
- Linux?  $\leadsto$  Just another copy of the same old stuff
- ...



New Operating Systems at SOSOP [9]

Aber dann...

*The Multicore Challenge!*





# Fallstudie: Dateideskriptortabelle in Linux

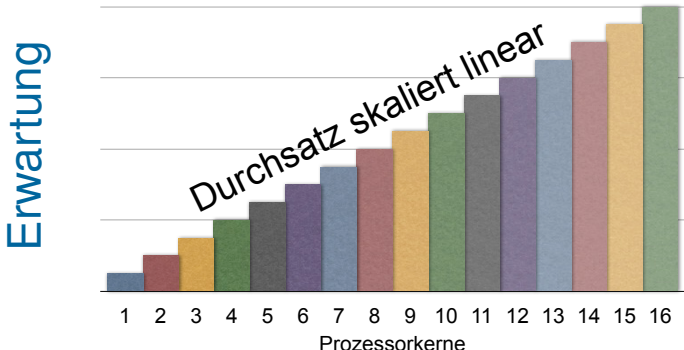
## ■ Boyd-Wickizer u. a. (OSDI 2008)

[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

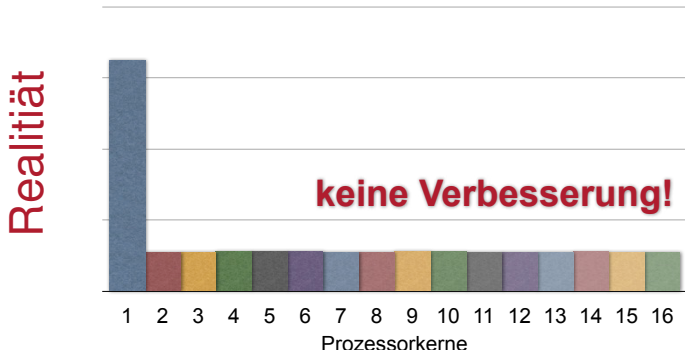


# Fallstudie: Dateideskriptortabelle in Linux

- 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



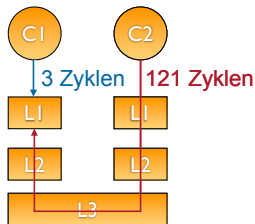
# Fallstudie: Dateideskriptortabelle in Linux

- 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
  1. Grobgranulares *Locking*  $\rightsquigarrow$  *false sharing*  $\rightsquigarrow$  keine Skalierbarkeit
  2. 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 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*  $\rightsquigarrow$  *false sharing*  $\rightsquigarrow$  keine Skalierbarkeit
  2. Geteilte Datenstruktur  $\rightsquigarrow$  *cache trashing*  $\rightsquigarrow$  Durchsatzabfall

**Multicore:** POSIX ( $\mapsto$  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: [15]
  - BS für 100 bis 1000 Kerne  $\rightsquigarrow$  *time sharing* wird zu *space sharing*
  - Letztlich ähnlicher Ansatz wie Barrelfish
- **TxOS** UT, SOSP 2009, Monolith (Linux): [11]
  - 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 Flaschenhälsen
    - 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.



**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]*

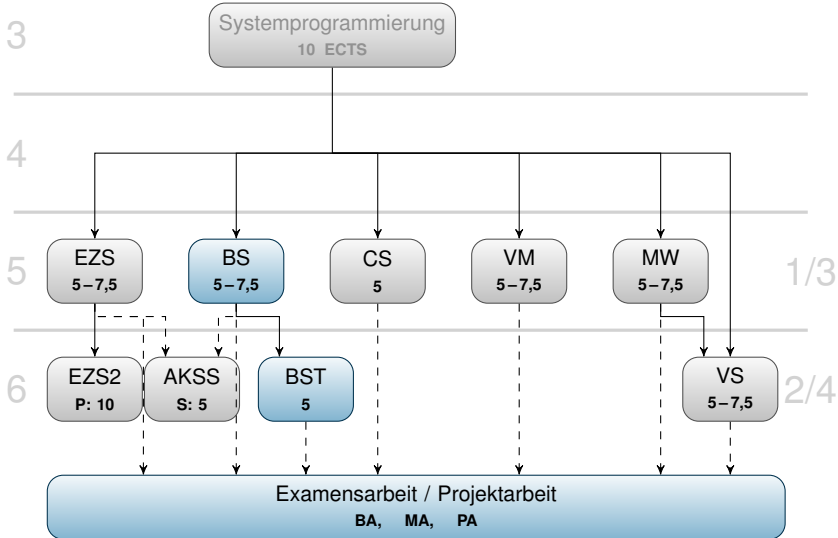
Fazit

# Es bleibt spannend!

Systementwurf für Skalierbarkeit ~ [CS] (WS 2021).







## Lernziele

---

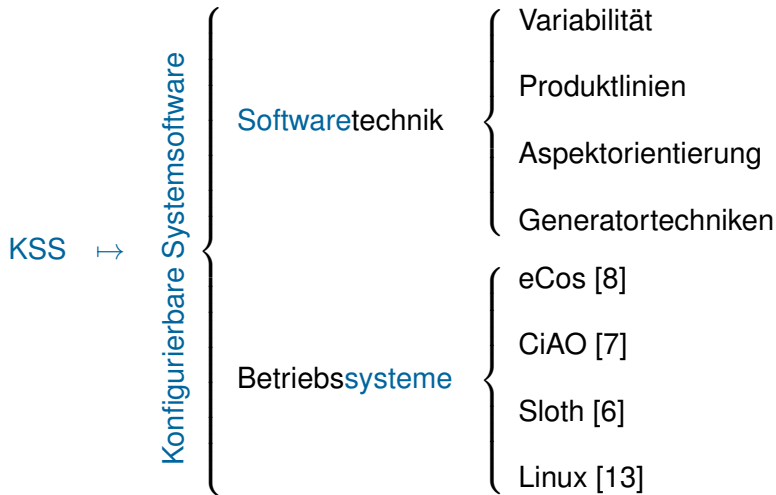
### Vorlesung

- *Wissen* zu Adressraumkonzepten von Betriebssystemen vertiefen
- *Verstehen* über (logische) Adressräume festigen
  - inhaltliches Begreifen verschiedener Facetten von Adressräumen
  - intellektuelle Erfassung des Zusammenhangs, in dem Adressräume stehen

### Übung $\rightsquigarrow$ mikrokern-ähnliches Betriebssystem

- *Anwenden* ausgewählter Vorlesungsinhalte für OOSTuBS
- *Analyse* der Anforderungen an und Gegebenheiten von OOSTuBS
- *Synthese* von Adressraumabstraktionen und OOSTuBS
- *Evaluation* des erweiterten OOSTuBS: Vorher-nachher-Vergleich





## Motivation: Special-Purpose Systems



## SLOTH: Threads as Interrupts

- **Idea: threads are interrupt handlers, synchronous thread activation is IRQ**
- Let interrupt subsystem do the scheduling and dispatching work
- Applicable to priority-based real-time systems
- Advantage: small, fast kernel with unified control-flow abstraction



### Spin-Locks brauchen ggf. viel Energie

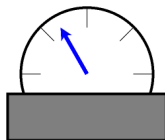
- Sleeping-Locks besser?
- Lock-free Algorithmen besser?
- Wait-free Algorithmen besser?
- ...



Problem: wie misst man den Energieverbrauch *einer* Applikation?

Zuweisung schwierig

- $P_{cpu} \Leftrightarrow$  App  
Scheduling, kritische Abschnitte, ...?
- $P_{mem} \Leftrightarrow$  App  
Kernel-Verwaltungsstrukturen, Buffer-Cache, ...?
- $P_{I/O} \Leftrightarrow$  App  
Nebenläufigkeit, Interrupts, ...?



### Stromerzeuger wollen gleichmäßigen Verbrauch

- hoher Strompreis, wenn viele Verbraucher
- niedriger (z.T. negativer!) Strompreis, wenn wenige Verbraucher

Daher:

=> Rechenzentren zum Ausgleichen

=> Rechen-Aufträge rund um den Globus verschicken

- „Strafe“ bei stark schwankendem Verbrauch

Daher:

=> BS soll Stromverbrauch „kappen“/regeln



Quelle: Wikipedia





Energieverbrauch:

$$E = \int_t P_t dt$$

Leistungsaufnahme:

$$P_t \propto V_t^2 f_t$$

Spannung  $V$  muss bei höherer Frequenz  $f$  höher sein.

Daher: statt einer CPU besser 2 CPUs mit halber Frequenz betreiben



### Linux:

handoptimiert (Code, Cache)

- - riesig
- - unübersichtlich
- - unwartbar
- ? (un)sicher
- + schnell

### „schönes“ OS:

„sauberer“ Code

- + klein
- + lesbar
- + wartbar
- + sicher
- - langsam

JITTY: Just-In-Time-compiliertes BS



### Wie alles anfang...

---

Gesucht: Code-Beispiele aus bekannten Betriebssystemen für Betriebssystem-Vorlesung



Problem:

**Linux:** Makro-/#ifdef-Hölle, selbst-modifizierender Code

**\*BSD:** viele Makros / viel #ifdef

**Minix:** Mikro-Kern

...: ...

**Windows:** keine Sourcen einsehbar



## Zur Zeit im Angebot:

- Bachelorarbeiten
- Masterarbeiten
- Projektarbeiten

<http://www4.informatik.uni-erlangen.de/DE/Theses/>

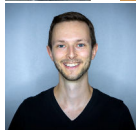
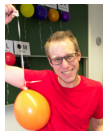
... oder persönlich nachfragen...!



# Das war's :-)

Das LS 4 BS-Team wünscht  
erfolgreiche und erholsame  
„Semesterferien“

... und ein Wiedersehen  
im Sommersemester 2021!





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.



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](https://www.usenix.org/legacy/event/osdi08/tech/full_papers/boyd-wickizer/boyd_wickizer.pdf).



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.





Austin T. Clements, M. Frans Kaashoek, Nikolai 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.



Sean Dorward, Rob Pike, Dave Presotto u. a. „The Inferno Operating System“. In: *Bell Labs Technical Journal* 2.1 (1997). URL: <http://www.vitanuova.com/inferno/papers/bltj.html>.



Wanja Hofer, Daniel Lohmann, Fabian Scheler u. a. „Sloth: Threads as Interrupts“. In: *Proceedings of the 30th IEEE International Symposium on Real-Time Systems (RTSS '09)*. (Washington, D.C., USA, 1.–4. Dez. 2009). IEEE Computer Society Press, Dez. 2009, S. 204–213. ISBN: 978-0-7695-3875-4. DOI: 10.1109/RTSS.2009.18.



Daniel Lohmann, Wanja Hofer, Wolfgang Schröder-Preikschat u. a. „CiAO: An Aspect-Oriented Operating-System Family for Resource-Constrained Embedded Systems“. In: *Proceedings of the 2009 USENIX Annual Technical Conference*. San Diego, CA, USA: USENIX Association, Juni 2009, S. 215–228. ISBN: 978-1-931971-68-3. URL: [https://www.usenix.org/legacy/event/usenix09/tech/full\\_papers/lohmann/lohmann.pdf](https://www.usenix.org/legacy/event/usenix09/tech/full_papers/lohmann/lohmann.pdf).





Daniel Lohmann, Fabian Scheler, Reinhard Tartler u. a. „A Quantitative Analysis of Aspects in the eCos Kernel“. In: *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (EuroSys '06)*. Hrsg. von Yolande Berbers und Willy Zwaenepoel. Leuven, Belgium: ACM Press, Apr. 2006, S. 191–204. ISBN: 1-59593-322-0. DOI: 10.1145/1218063.1217954.



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>.



Rob Pike. *Systems Software Research is Irrelevant*. Talk. CS Colloquium, Columbia University. URL: <http://herpolhode.com/rob/utah2000.pdf> (besucht am 09.12.2010).



Rob Pike, Dave Presotto, Sean Dorward u. a. „Plan 9 from Bell Labs“. In: *Computing Systems 8.3* (1995), S. 221–254.



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.







*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.



Wolfgang Schröder-Preikschat. *Betriebssystemtechnik*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4, 2015 (jährlich). URL: [https://www4.cs.fau.de/Lehre/SS15/V\\_BST](https://www4.cs.fau.de/Lehre/SS15/V_BST).



Wolfgang Schröder-Preikschat. *Concurrent Systems*. Vorlesung mit Übung. Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik 4, 2015 (jährlich). URL: [https://www4.cs.fau.de/Lehre/WS15/V\\_CS](https://www4.cs.fau.de/Lehre/WS15/V_CS).



Reinhard Tartler, Daniel Lohmann, Julio Sincero u. a. „Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem“. In: *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2011 (EuroSys '11)*. (Salzburg, Austria). Hrsg. von Christoph M. Kirsch und Gernot Heiser. New York, NY, USA: ACM Press, Apr. 2011, S. 47–60. ISBN: 978-1-4503-0634-8. DOI: 10.1145/1966445.1966451.



Linus Torvalds und David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. HarperCollins, 2001. ISBN: 978-0066620725.





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.

