

# Betriebssystemtechnik

*Adressräume: Trennung, Zugriff, Schutz*

## I. Einleitung

Wolfgang Schröder-Preikschat / Volkmar Sieh

19. April 2023



## Einführung

Motivation

Grundlagen

Inhalt

## Organisation

Voraussetzungen

Veranstaltungsbetrieb

Leistungsnachweise

## Anhang



~ **in räumlicher Hinsicht**  $\rightsquigarrow$  BST

- **Angriffssicherheit** (*security*)

- Schutz einer Entität vor seiner Umgebung
- Immunität
- verhindern, in einen Adressraum einbrechen zu können

- **Betriebssicherheit** (*safety*)

- Schutz der Umgebung vor einer Entität
- Isolation
- verhindern, aus einem Adressraum ausbrechen zu können

~ **in zeitlicher Hinsicht**  $\rightsquigarrow$  EZS[3]

- Einhaltung von Terminen, Vermeidung von Interferenzen

~ **in energetischer Hinsicht**

- Abfederung von Energiespitzen, Einhaltung von Energiebudgets





- Linux-binär-kompatibler Betriebssystemkern
- programmiert von Mitarbeitern und Studentinnen und Studenten
- „sauberer“ Source-Code
- nur ca. 150k LOC (inkl. vielen Kommentaren)
- Code für Lehrstuhlprojekte
- Beispiel-Code für BS und **BST**
- <https://gitlab.cs.fau.de/i4/jitty/jitty-os.git>



Definitionen:

Ablaufumgebung (Task):

Menge von Code, Daten, offene Dateien, usw.

Aktivitätsträger (Thread):

Einheit, die Code ausführen kann

typischer Unix Prozess:

*eine* Ablaufumgebung mit *einem* Aktivitätsträger

hier: Prozess:

*eine* Ablaufumgebung mit *ggf. mehreren* Aktivitätsträgern

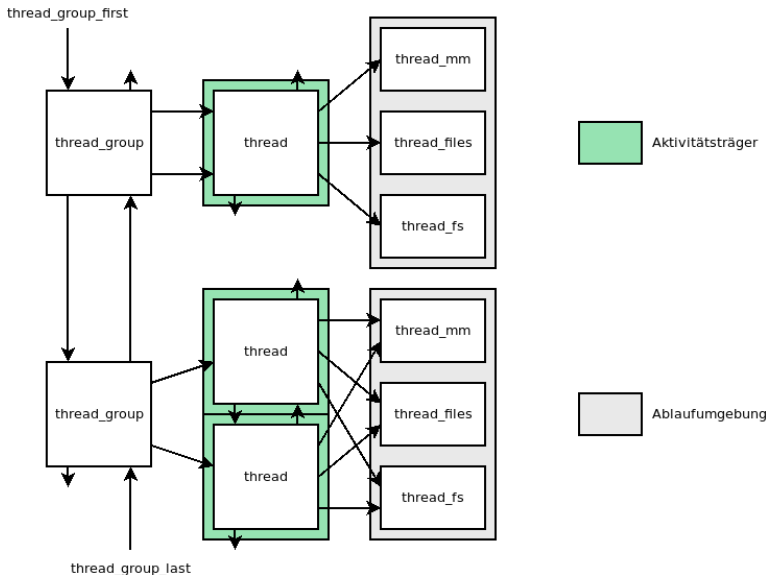


## Feingranulare Unterteilung der Ablaufumgebung:

```
1 struct thread {
2     struct thread *prev;
3     struct thread *next;
4     ...
5     struct thread_mm *mm;
6     struct thread_files *files;
7     struct thread_fs *fs;
8     ...
9     long reg[8];
10    uint8_t stack[16*1024];
11 };
12
13 struct thread_group {
14     struct thread_group *prev;
15     struct thread_group *next;
16     ...
17     struct thread_group *parent;
18     struct thread_group *child_first;
19     struct thread_group *child_next;
20     ...
21     struct thread *first;
22     struct thread *last;
23 };
1 struct thread_mm {
2     ...
3     struct mm_reg *reg_first;
4     struct mm_reg *reg_last;
5 };
6
7 struct thread_files {
8     ...
9     struct file *file[256];
10 };
11
12 struct thread_fs {
13     ...
14     struct dentry *root;
15     struct dentry *cwd;
16 };
17
18 struct thread_group *thread_group_first;
19 struct thread_group *thread_group_last;
```



# Prozess – Beispiel JITTY



## ■ realer ~

- reflektiert die phys(ikal)ischen Eigenschaften des Rechensystems
- nicht zu jeder Adresse gibt es einen Adressaten (Speicher, Geräte)
- die Bindung zwischen beiden ist fest, zur Laufzeit unveränderlich
  - **Vorsicht:** Speicherbankumschaltung oder PCI-Config-Space
- ungültige Adressen implizieren undefiniertes/fehlerhaftes Verhalten
  - **Vorsicht:** Probing von Geräten beim Booten

## ■ logischer ~

- reflektiert die strukturellen Eigenschaften eines Programms
- zu jeder Adresse gibt es immer einen (speicher-) residenten Adressaten
- die Bindung zwischen beiden ist jedoch lose, zur Laufzeit veränderlich
  - *Beispiele:* brk, sbrk, mmap, munmap, malloc/free, Stack
- ungültige Adressen – innerhalb des Adressraums – gibt es nicht
  - **Vorsicht:** Unterschied zwischen Segmentierung und Seitennummerierung

## ■ virtueller ~

- reflektiert die gegenwärtige/zukünftige Auslastung des Rechensystems
- zu einer Adresse kann es zeitweilig einen nichtresidenten Adressaten geben
- ansonsten „erben“ die Adressen alle Eigenschaften logischer Adressräume





Beispiel Intel:

Segment-Nummer und Offset



lineare Adresse



physikalische Adresse

virtuelle Adresse?



Beispiel IBM-AT:

0x00000000	0x0009ffff	DRAM
0x000a0000	0x000b7fff	reserviert für VGA-Grafik
0x000b8000	0x000bffff	VGA-Text
0x000c0000	0x000cffff	VGA-BIOS
0x000d0000	0x000dffff	reserviert für z.B. BIOS-Ext.
0x000e0000	0x000fffff	BIOS
0x00100000	0x00efffff	DRAM
0x00f00000	0x00ffffff	reserviert für ISA-I/O
0x01000000	?	DRAM
0xfffe0000	0xffffffff	BIOS

Was passiert bei Zugriff auf nicht zugewiesene Speicherbereiche?



Segmentierung oder Eingrenzung von Programmen:

- **Maschinenprogrammenebene** (Ebene 3)
  - der **Prozessor** ermöglicht Immunität/Isolation in Hardware
    - MMU (Abk. *memory management unit*) ..... logischer Adressraum
    - MPU (Abk. *memory protection unit*) ..... phys(ikal)ischer Adressraum
  - das **Betriebssystem** programmiert diese Hardware problemspezifisch
- **Programmiersprachenebene** (Ebene 5)
  - der **Compiler** ermöglicht Immunität/Isolation in Software
  - Programme liegen in einer **typischeren Programmiersprache** vor

Prozesse können die durch ihren logischen Adressraum jew. definierte **Schutzdomäne** nicht oder nur kontrolliert verlassen

- Abwesenheit von Prozessor- und Speicherfehlern vorausgesetzt
  - je nach Abstraktionsebene aber mit unterschiedlichem Wirkungsfeld

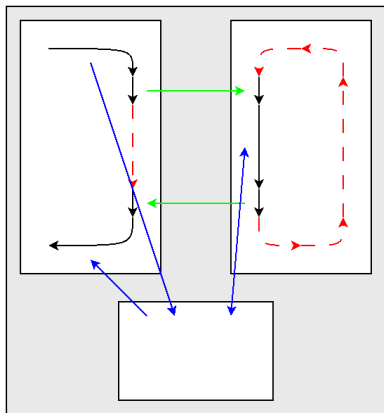
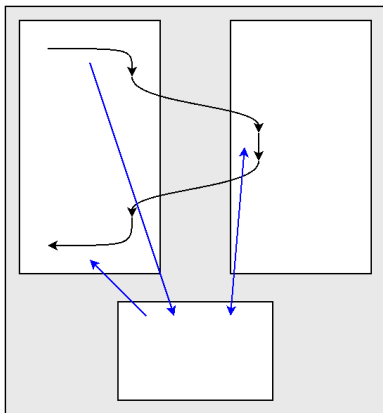


Grenzen überschreitende Operationen als Folge der Trennung

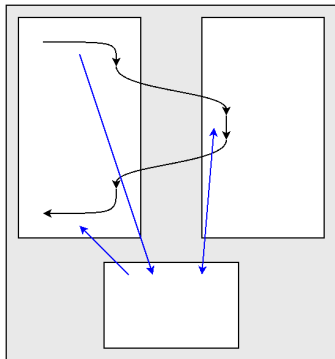
- durch **Wechsel** der Schutzdomäne
  - prozedurbasierte Technik
    - Systemaufruf, leichtgewichtiger Fernaufruf
    - Kontrollflussfortsetzung im anderen Adressraum
  - koroutinenbasierte Technik
    - Nachrichtenversenden, Fernaufruf
    - Kontrollflusswechsel hin zum anderen Adressraum
- durch **Mitbenutzung** (*sharing*) von Adressraumbereichen
  - Datenverbund (*data sharing*)
    - mit gleichförmigen oder ungleichförmigen Lese-/Schreibrechten
  - Gemeinschaftsbibliothek (*shared library*)
- durch **Kombinierung** beider Ansätze
  - Einrichtung eines Datenverbunds beim Wechsel der Schutzdomäne
    - aus dem „eingewechselten Adressraum“ heraus veranlasst
    - zum Lesen/Schreiben von Entitäten des „ausgewechselten Adressraums“



## Grenzen überschreitende Operationen als Folge der Trennung



## Grenzen überschreitende Operationen als Folge der Trennung



### Vorteile:

- keine unnötigen Threads/Kontextwechsel

### Nachteile:

- ggf. Verlust von Cache-Inhalten
- Kontextwechsel über BS

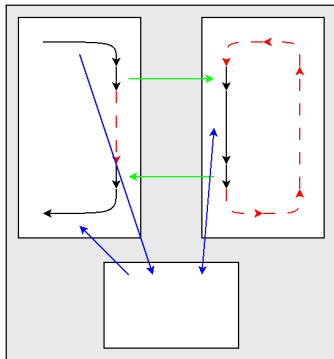
## Grenzen überschreitende Operationen als Folge der Trennung

### Vorteile:

- Cache-Inhalte bleiben erhalten

### Nachteile:

- mehr Threads/Kontextwechsel
- send/recv über BS



- **hardwarebasiert**, segment- oder seitenorientiert
  - MMU/MPU vergleicht Zugriffsart und Zugriffsrecht
    - Lesen, Schreiben, Ausführen – auch in Kombination
  - CPU begeht Ausnahme von normaler Programmausführung
    - lässt den aktuellen Maschinenbefehl in die Falle (*trap*, *exception*) laufen
    - bewirkt damit eine **synchrone Programmunterbrechung**
  - Betriebssystem führt Ausnahmebehandlung [2] durch
    - Wiederaufnahmemodell: Fortsetzung des unterbrochenen Prozesses
    - Beendigungsmodell: Abbruch des unterbrochenen Prozesses
- **softwarebasiert**, datentyporientiert ⇒ **sprachbasiert**
  - Laufzeitsystem – d. h., der Prozess selbst – führt o. g. Funktionen durch
  - bestimmte Überprüfungen nimmt jedoch bereits der (JIT-) Compiler vor
    - alle statisch, also *vor* Laufzeit, entscheidbaren Zugriffsoperationen
- in Synergie beider Ansätze: **Befähigung** (*capability*, [1])
  - befähigungsbasierte Systeme sind kompliziert – obwohl ideal zum Schutz





## 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 StuBS
- *Analyse* der Anforderungen an und Gegebenheiten von StuBS
- *Synthese* von Adressraumabstraktionen und StuBS
- *Evaluation* des erweiterten  $\text{StuBS}_{ml}$ : Vorher-nachher-Vergleich



- Einführung: vertikale (räumliche) Isolation ÜV
  
- Einflussfaktoren
  - Systemaufrufe: Befehlsformate der Maschinenprogrammebene ÜV
  - Betriebssystemarchitektur: {Nano,Mikro,Makro,Exo}kern V
  - Hierarchien: Schichtenstruktur von Betriebssystemen V
  
- Adressraumkonzepte
  - Seitenadressierung: ein-/mehrstufig, invertiert, spärlich ÜV
  - Segmentadressierung: pur, seitenbasierte Hybride V

... jeweils mit Beispielen aus der Praxis.



- Sprachbasierung: Systemprogrammiersprachen S
- Adressraummodelle
  - Mehradressraumsystem: total/partiell privat ÜV
  - Einadressraumsystem: Randomisierung, Befähigungen V
- Spezialfälle
  - adaptiver Hauptspeicherschutz: TLB, typsichere Programme ÜV
  - virtuell gemeinsamer Speicher: seitenbasierte „IPC“ ÜV
  - virtuell nicht-flüchtiger Hauptspeicher: NVRAM, Transparenz V
  - dynamisches Binden: Gemeinschaftsbibliotheken V
- Nachlese, Ausblick V

... jeweils mit Beispielen aus der Praxis.



## Einführung

Motivation

Grundlagen

Inhalt

## Organisation

Voraussetzungen

Veranstaltungsbetrieb

Leistungsnachweise

## Anhang



## ■ Voraussetzung

### Softwaresysteme

- SP, SPiC
- BS  $\Leftrightarrow$  StuBS

### Programmiersysteme

- C, C++, make
- ASM

### Hardwaresysteme

- x86/x86\_64
- Mehrkerner

## ■ Erfahrung

- in der hardwarenahen Programmierung
  - Gerätetreiber, Unterbrechungsbehandlung, Prozesswechsel
- in der Fehlersuche/-beseitigung (*debugging*) in Betriebssystemen
  - nichtsequentielle Programme, mehrkernige Prozessoren
- in der projektorientierten Entwicklung nativer Systemprogramme

## ■ Erwartung

- intrinsische Motivation, kritisches Denken, positive Fehlerkultur



- Vorlesungs-, Übungs- und Rechnerzeiten:
  - auf `sys.cs.fau.de` dem Reiter „Lehre“ folgen

- Unterrichtssprache:



- Vorlesung und Übung



- Fachbegriffe

- informatische Fachsprache
  - Sachwortverzeichnis (in Arbeit und Überarbeitung)
    - <https://www4.cs.fau.de/DE/~wosch/glossar.pdf>

- **Tafelübung**  $\leadsto$  „*learning by exploring*“
  - Anmeldung über [WAFFEL](#)<sup>1</sup> (URL siehe Leitseite von BST)
  - Übungsaufgaben sind (bevorzugt) in Gruppen zu bearbeiten
- **Rechnerarbeit**  $\leadsto$  „*learning by doing*“, **kein Tafelersatz**
  - Anmeldung ist nicht vorgesehen, reservierte Arbeitsplätze s.o.
  - bei Fragen zu den Übungsaufgaben, Übungsleiter konsultieren
    - Email senden bzw. einfach vorbeischaun. . .

*Der, die, das.  
Wer, wie, was?  
Wieso, weshalb, warum?  
Wer nicht fragt, bleibt dumm!*



<sup>1</sup>Abk. für Webanmeldefrickelformular Enterprise Logic

## 5 ECTS

- BST und OOSTuBS<sub>ml</sub>
- 4 SWS (2 V + 2 Ü)

- Prüfungsgespräch
  - 30 Minuten
  - Stoff zu V + Ü

## 7,5 ECTS

- BST und MPStuBS<sub>ml</sub>
- 6 SWS (2 V + 2 Ü + 2 EÜ)
  - erweiterte Übung
  - ggf. auch Extraaufgabe

- Prüfungsgespräch
  - 30 Minuten
  - Stoff zu V + Ü + EÜ

- erfolgreiche Bearbeitung der Übungsaufgaben ist Voraussetzung
- Anmeldung zum Prüfungsgespräch per *E-Mail* an:  
[sieh@cs.fau.de](mailto:sieh@cs.fau.de)
  - angeben ob 5 oder 7,5 ECTS
  - Termin oder Terminfenster mitsenden
    - Prüfungszeitraum (Ausnahmen bestätigen die Regel)





## Einführung

Motivation

Grundlagen

Inhalt

## Organisation

Voraussetzungen

Veranstaltungsbetrieb

Leistungsnachweise

## Anhang





- Volkmar Sieh, Dr.-Ing.
  - Vorlesung
  - [sys.cs.fau.de/person/sieh](https://sys.cs.fau.de/person/sieh)



- Wolfgang Schröder-Preikschat, Prof. Dr.-Ing. habil.
  - Vorlesung (ggf. Vertretung)
  - [sys.cs.fau.de/person/wosch](https://sys.cs.fau.de/person/wosch)



- Phillip Raffeck, M. Sc.
  - Übungen
  - [sys.cs.fau.de/person/raffeck](https://sys.cs.fau.de/person/raffeck)



- Dustin Nguyen, M. Sc.
  - Übungen
  - [sys.cs.fau.de/person/nguyen](https://sys.cs.fau.de/person/nguyen)



- [1] DENNIS, J. B. ; HORN, E. C. V.:  
Programming Semantics for Multiprogrammed Computations.  
In: *Communications of the ACM* 9 (1966), März, Nr. 3, S. 143–155
- [2] GOODENOUGH, J. B.:  
Exception Handling: Issues and a Proposed Notation.  
In: *Communications of the ACM* 18 (1975), Nr. 12, S. 683–696
- [3] SCHRÖDER-PREIKSCHAT, W. :  
*Echtzeitsysteme*.  
[http://www4.informatik.uni-erlangen.de/Lehre/WS05/V\\_EZS](http://www4.informatik.uni-erlangen.de/Lehre/WS05/V_EZS), 2005 ff.
- [4] SCHRÖDER-PREIKSCHAT, W. ; KLEINÖDER, J. :  
*Systemprogrammierung*.  
[http://www4.informatik.uni-erlangen.de/Lehre/WS08/V\\_SP](http://www4.informatik.uni-erlangen.de/Lehre/WS08/V_SP), 2008 ff.

