Betriebssystembau (BSB)

VL 6 – Unterbrechnungssynchronisation in der Realität

Software-Interrupts

Alexander Krause

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

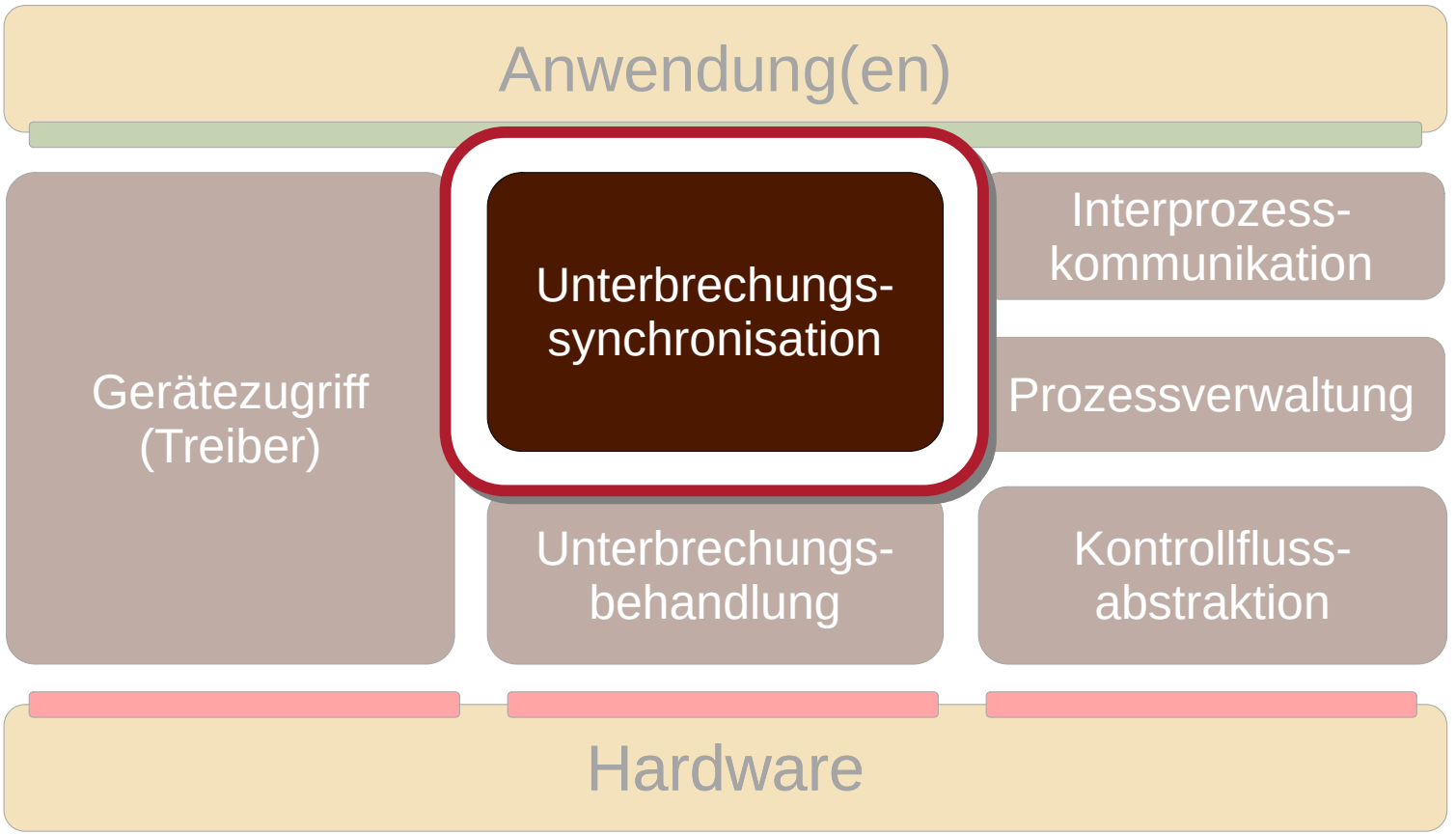
`https://sys.cs.tu-dortmund.de/de/lehre/ws25/bsb`

WS 25 – 11. November 2025

technische universität dortmund

arbeitsgruppe systemsoftware

Überblick: Einordnung dieser VL



Betriebssystementwicklung

Agenda

Hardware-Unterbrechungen

Probleme:

- Während der ganzen Unterbrechungslaufzeit sind alle weiteren bzw. alle niederprioren Unterbrechungen gesperrt.
  - Gefahr von großer Interrupt-Verzögerung
  - Gefahr von Datenverlusten
- Unterbrechungsbehandlung kann nicht passiv warten.
  - Interrupts sind gesperrt

☞ Sperrzeiten minimieren

Aufteilung Unterbrechungsbehandlung

Unterbrechungsbehandlung zweigeteilt

Aufteilung Unterbrechungsbehandlung

Unterbrechungsbehandlung zweigeteilt

1. **Teil:** Befriedigt die Hardware
   - Liest Zeichen/Pakete/... der Hardware aus und kopiert in einen Puffer.
   - Interagiert nur minimal mit dem Rest des Systems.
   - Kann (fast) immer ablaufen.

Aufteilung Unterbrechungsbehandlung

☞ Unterbrechungsbehandlung zweigeteilt

1. **Teil:** Befriedigt die Hardware

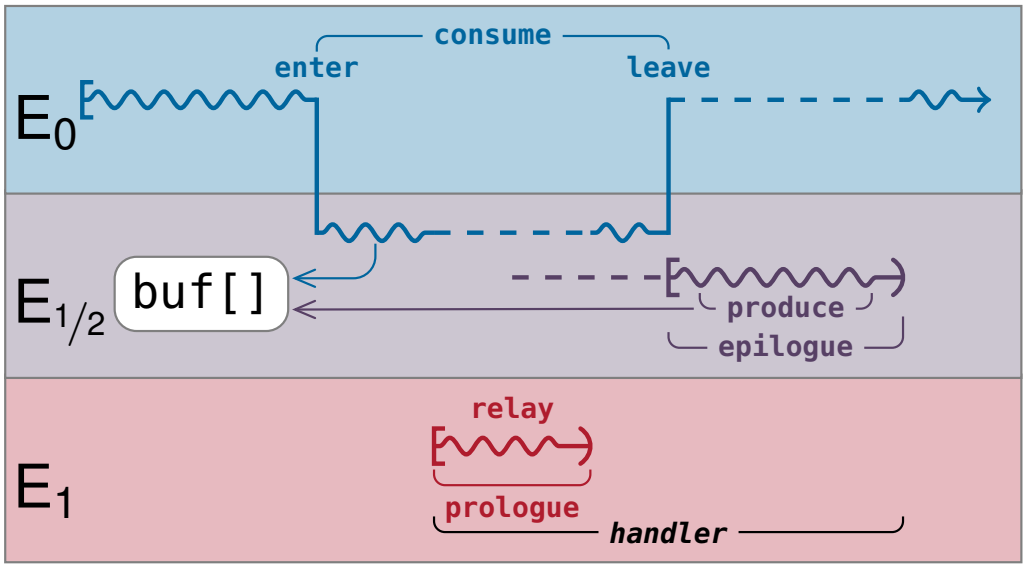
   - Liest Zeichen/Pakete/... der Hardware aus und kopiert in einen Puffer.
   - Interagiert nur minimal mit dem Rest des Systems.
   - Kann (fast) immer ablaufen.
     Gegebenenfalls leer! Beispiel: Timer/Uhr-Interrupt

Aufteilung Unterbrechungsbehandlung

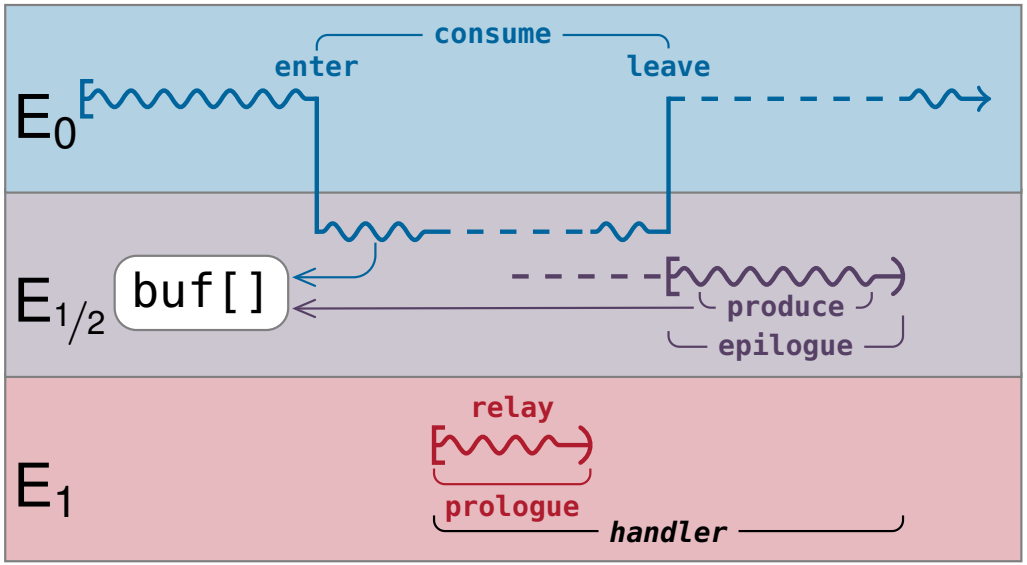
☞ Unterbrechungsbehandlung zweigeteilt

1. **Teil:** Befriedigt die Hardware
   - Liest Zeichen/Pakete/... der Hardware aus und kopiert in einen Puffer.
   - Interagiert nur minimal mit dem Rest des Systems.
   - Kann (fast) immer ablaufen.
     Gegebenenfalls leer! Beispiel: Timer/Uhr-Interrupt
2. **Teil:** Hardwareunabhängig
   - Liest Zeichen/Pakete/... aus Puffer aus und verarbeitet sie weiter.
   - Ist weitgehend Hardware-unabhängig.
   - Kann (fast) immer unterbrochen werden.

Pro-Epilog-Modell – Wdh.

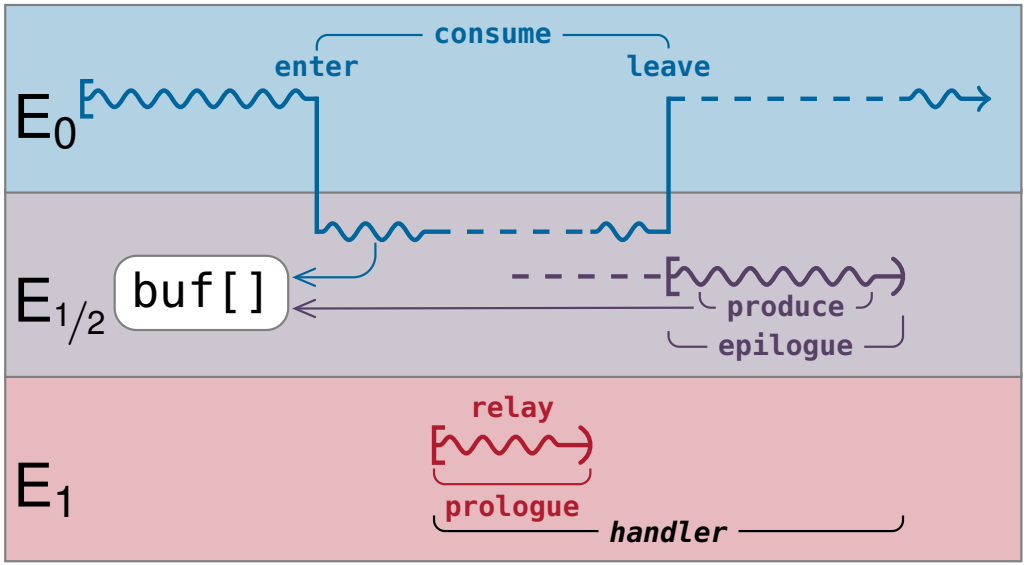


Pro-Epilog-Modell – Wdh.



- **Prolog**
  - läuft auf Hardwareunterbrechungsebene
  - ist **kurz**, fasst wenig oder gar keinen Zustand an
  - kopiert z. B. Daten vom Gerät in einen Puffer

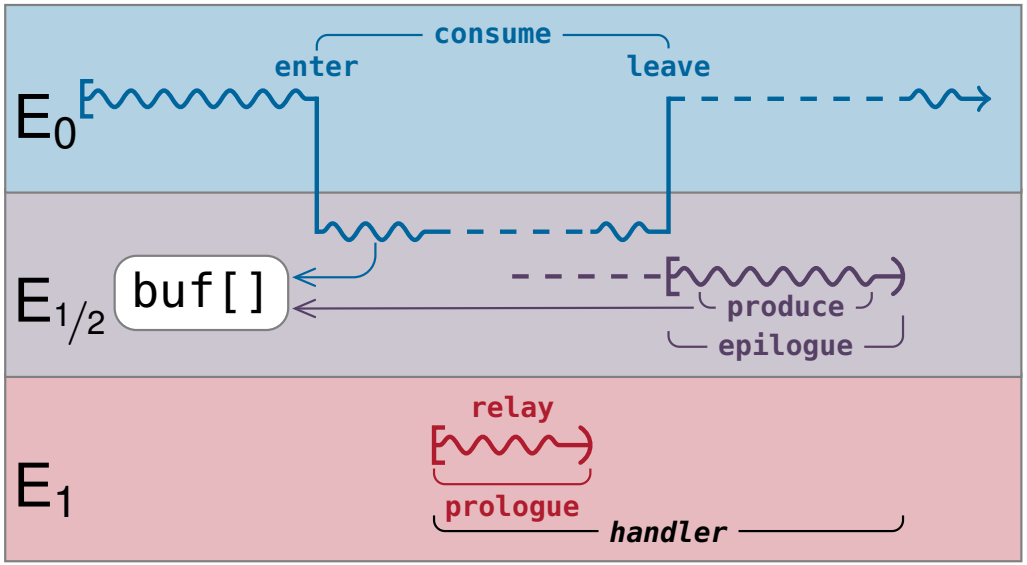
Pro-Epilog-Modell – Wdh.



- **Prolog**
  - läuft auf Hardwareunterbrechungsebene
  - ist **kurz**, fasst wenig oder gar keinen Zustand an
  - kopiert z. B. Daten vom Gerät in einen Puffer
- **Epilog**
  - läuft auf Epilogebene $E_{1/2}$
  - hat Zugriff auf größten Teil des Zustands
  - erledigt die eigentliche Arbeit

(Vgl. Vl. 6 Folie 34ff.)

Pro-Epilog-Modell – Wdh.



- **Operationen**
  - `enter()` betritt E.-Ebene
  - `leave()` verlässt E.-Ebene
  - `relay()` fordert Epilog an

- **Prolog**
  - läuft auf Hardwareunterbrechungsebene
  - ist **kurz**, fasst wenig oder gar keinen Zustand an
  - kopiert z. B. Daten vom Gerät in einen Puffer
- **Epilog**
  - läuft auf Epilogebene $E_{1/2}$
  - hat Zugriff auf größten Teil des Zustands
  - erledigt die eigentliche Arbeit

(Vgl. Vl. 6 Folie 34ff.)

Agenda

Top/Bottom Halves 1/2

Although it is not always clear how to divide the work between the top and bottom half, a couple of useful tips help:

- If the work is time sensitive, perform it in the interrupt handler.
- If the work is related to the hardware, perform it in the interrupt handler.
- If the work needs to ensure that another interrupt (particularly the same interrupt) does not interrupt it, perform it in the interrupt handler.
- For everything else, consider performing the work in the bottom half.

(*Linux Kernel Development* [2, Seite 134])

Top/Bottom Halves 2/2

- Entspricht dem Pro-Epilog-Modell unter Linux [2]
- Sicht auf den Ablauf ist invers zu BSB: Unterbrechungsebene ist „oben“
  - *top half* $\rightsquigarrow$ Prolog
  - *bottom half* $\rightsquigarrow$ Epilog
- Stellt `statisch` 32 *bottom halfs* bereit
- Abarbeitung geschieht bei der Rückkehr von der Unterbrechungsebene
- *bottom half* laufen mit aktivierten Unterbrechungen
- Wurde in Linux 2.5 entfernt

SoftIRQ

- *SoftIRQ* [2] entspricht Epilog
- Von 1 bis 32 durchnummeriert; entspricht der Priorität
- Gleicher SoftIRQ darf mehrfach auf verschiedenen Prozessoren laufen
  ⇝ sollte nur Prozessor-lokale Daten modifizieren
- Wird nur von einer Unterbrechungsbehandlung unterbrochen
- Ausführung erfolgt ...
  - bei der Rückkehr aus der Unterbrechungsbehandlung
  - auf dem Code-Pfad, der explizit auf SoftIRQs prüft
  - in separatem Kernel-Thread mit Namen „*ksoftirq/X*“ (einer pro Kern)
- `while (pending) { pending = false; handler(); }`-Schleife
- Aktuell (v6.6) gibt es 10 SoftIRQs
- Auslösen mittels `raise_softirq()`

Tasklets

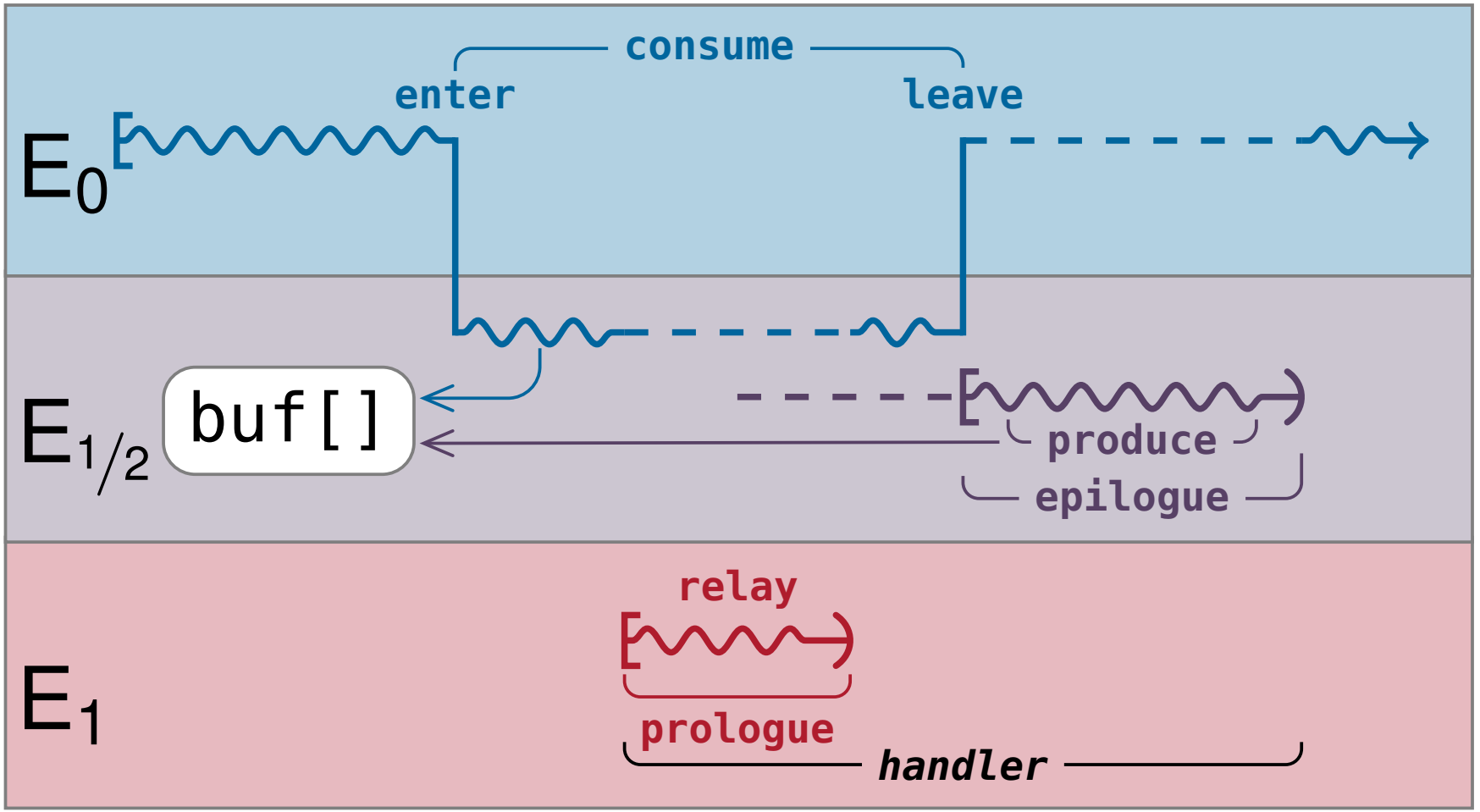
- *tasklets* [2] entsprechen Epilog (*bottom half*)
- Ist der bevorzugte Weg zur Realisierung eines Epilogs
- Liste von abzuarbeitenden Epilogen
- Ein Tasklet darf gleichzeitig nur auf einem Prozessor laufen
- Baut auf SoftIRQ auf $\rightsquigarrow$ läuft daher in einem eigenen SoftIRQ
- Gibt hochpriore (*HI_SOFTIRQ*) und niedrigpriore (*TASKLET_SOFTIRQ*) Tasklets

Work Queues

- *work*-Datenstruktur [2] entsprecht Epilog (*bottom half*)
- Eine Instanz von *work* ...
  - kann auf irgendeinem oder einem dezidierten Kern ausgeführt werden
  - darf sich schlafen legen
- Eine Instanz von *work queue* ...
  - beinhaltet einen *kernel thread* pro Kern
  - arbeitet Liste von Aufträgen (*work*) ab
  - führt Instanz *work* in einem Prozesskontext aus
- Bei Bedarf eigene Instanz von *work queue* erzeugbar

Welche Epilog-Variante ist „besser“? 1/2

- Was passiert, wenn der Epilog ein Lock holt?



Welche Epilog-Variante ist „besser“? 2/2

Welche Epilog-Variante ist „besser“? 2/2

- Es kommt drauf an ...

Welche Epilog-Variante ist „besser“? 2/2

- Es kommt drauf an ...
  - Möchte ich bloß Arbeit in einem Treiber verzögern? ⇝ *tasklet*

Welche Epilog-Variante ist „besser“? 2/2

- Es kommt drauf an ...
  - Möchte ich bloß Arbeit in einem Treiber verzögern? ⇝ *tasklet*
  - Benötige ich zusätzlich noch ein passiv-wartendes Lock? ⇝ *work queue*

Welche Epilog-Variante ist „besser“? 2/2

- Es kommt drauf an ...
  - Möchte ich bloß Arbeit in einem Treiber verzögern? ⇝ *tasklet*
  - Benötige ich zusätzlich noch ein passiv-wartendes Lock? ⇝ *work queue*
  - Möchte ich etwas `wirklich` hochfrequent erledigen? ⇝ *softirq*

(Angaben ohne Gewähr)

Welche Epilog-Variante ist „besser“? 2/2

- Es kommt drauf an ...
  - Möchte ich bloß Arbeit in einem Treiber verzögern? ⇝ *tasklet*
  - Benötige ich zusätzlich noch ein passiv-wartendes Lock? ⇝ *work queue*
  - Möchte ich etwas `wirklich` hochfrequent erledigen? ⇝ *softirq*

  (Angaben ohne Gewähr)
- Weitere Details liefert z. B. „*Linux Kernel Development*“oder „*Understanding The Linux Kernel*“ [1]

Epilog	Kontext	Sequentialisierung
SoftIRQ	Unterbrechung	Keine
Tasklet	Unterbrechung	Gegen das gleiche Tasklet
Work Queues	Prozess	Keine

(Tabelle 8.3 aus „*Linux Kernel Development*“ [2, Seite 156])

Welche Epilog-Variante ist „besser“? 2/2

- Es kommt drauf an ...
  - Möchte ich bloß Arbeit in einem Treiber verzögern? ⇝ *tasklet*
  - Benötige ich zusätzlich noch ein passiv-wartendes Lock? ⇝ *work queue*
  - Möchte ich etwas `wirklich` hochfrequent erledigen? ⇝ *softirq*

  (Angaben ohne Gewähr)
- Weitere Details liefert z. B. „*Linux Kernel Development*“oder „*Understanding The Linux Kernel*“ [1]

Epilog	Kontext	Sequentialisierung
SoftIRQ	Unterbrechung	Keine
Tasklet	Unterbrechung	Gegen das gleiche Tasklet
Work Queues	Prozess	Keine

(Tabelle 8.3 aus „*Linux Kernel Development*“ [2, Seite 156])

- **Empfehlung:** Bitte passiv programmieren. Nicht zu stark in das System eingreifen. ⇝ schwächste Variante wählen!

Agenda

Zusammenfassung

- Auch in der Realität gibt es ein Pro-Epilog-Modell
- Es heißt bloß anders (*top half*, *bottom half*)
- Viele unterschiedliche Möglichkeiten, einen Epilog in Linux zu realisieren.
- Ziel ist (fast) immer: Eine schnelle Reaktion auf (wichtige/hochfrequente) Ereignisse
- Epiloge sind „Unterbrechungen” in Software.
- In anderen Betriebssystemen gibt es ähnliches Konzepte.

Referenzen

[1] Daniel Pierre Bovet und Marco Cesati. *Understanding The Linux Kernel*. 3rd. O'Reilly Media Inc., Nov. 2005. isbn: 0596005652.

[2] Robert Love. *Linux Kernel Development*. 3rd. Boston, MA, USA: Addison-Wesley, Juni 2010. isbn: 9780672329463.