Grundlagen der Rechnerarchitektur und Betriebssysteme

Traps

Florian Knoch · Lehrstuhl für Praktische Informatik,
insbes. Systemnahe Programmierung, Universität Bamberg

Unterbrechungen

Unterbrechungen

Terminologie

Ereignis	*Interrupt*	*Exception*	*Trap*
Division durch Null			
Systemaufruf (en. *System Call*)			
Speicherzugriffsfehler (en. *Page Fault*)			
Tastendruck			
Signal des Taktgebers (en. *Timer*)			
Ankunft eines Netzwerkpakets			

AUFGABE 1

Welche dieser Ereignisse lösen *Interrupts*, *Exceptions* oder *Traps* aus? In welchem Zusammenhang stehen diese Begriffe?

Unterbrechungen

Terminologie

Ereignis	*Interrupt*	*Exception*	*Trap*
Division durch Null		✓	✓
Systemaufruf (en. *System Call*)		✓	✓
Speicherzugriffsfehler (en. *Page Fault*)		✓	✓
Tastendruck	✓		✓
Signal des Taktgebers (en. *Timer*)	✓		✓
Ankunft eines Netzwerkpakets	✓		✓

Unterbrechungen

Anwendungen

AUFGABE 2

Welche Rolle übernehmen Unterbrechungen im Betriebssystem? Nennen Sie drei Beispiele.

Unterbrechungen

Anwendungen

AUFGABE 2

Welche Rolle übernehmen Unterbrechungen im Betriebssystem? Nennen Sie drei Beispiele.

- **Ein- und Ausgabe**: interrupt-getriebene Verarbeitung von Ein- und Ausgabesignalen zur besseren Auslastung der CPU
- **Multitasking**: Rückgabe der Kontrolle von Prozessen an das Betriebssystem
- **Energiemanagement**: Rückkehr aus stromsparenden Gerätezuständen mittels Hardware-Interrupt

Beispiel: Energiemanagement

Arduino Uno R3 (ATmega328P) [1]

- zahlreiche verschiedene **Schlafmodi** (Idle, ADC Noise Reduction, Power Down, Power Save, (Extended) Standby)
- deaktivieren Komponenten (z. B. CPU, I/O, Analog-Digital-Konverter, Timer, ...) in unterschiedlichem Umfang
- **Aufwachen** je nach Modus durch Timer, I/O, ...

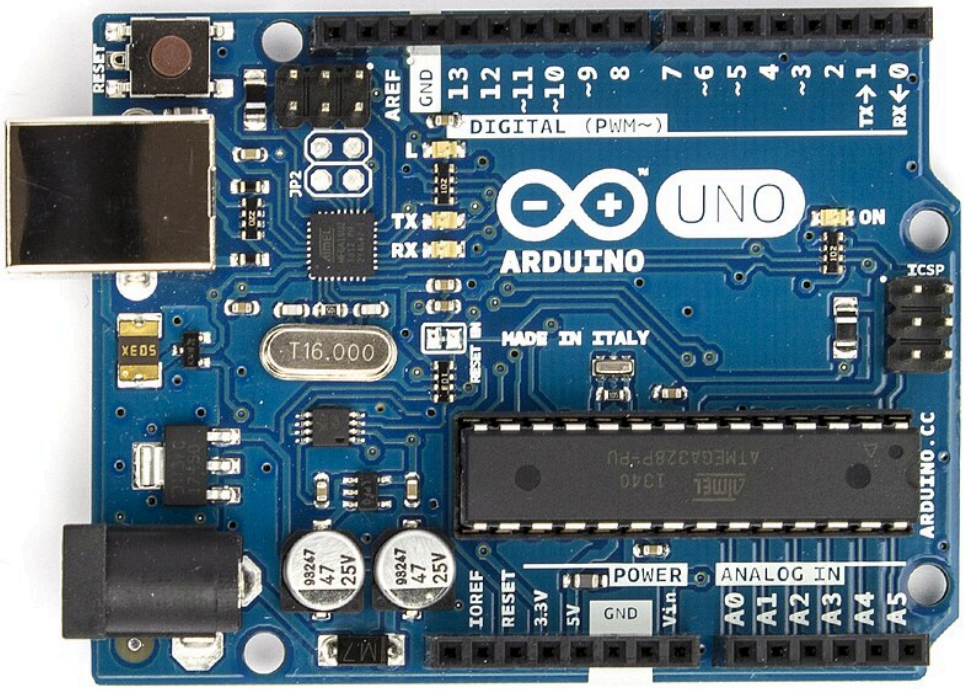

Abbildung 1: Arduino Uno R3

Kontextwechsel

ANWENDUNGSBEISPIEL

Start des Betriebssystems

- Beispiel: **simples Betriebssystems für RISC-V**
- beim Starten werden zunächst einige Einstellungen getroffen
- entscheidend für Kontextwechsel: *Trap Handler*
- danach Übergabe ans User-Programm

KERNEL.C

```
extern void trap(void);

void setup(void) {
  // Privilegienmodi konfigurieren
  // Interrupts aktivieren
  // Startpunkt festlegen
  // …

  // Trap Handler hinterlegen
  w_mtvec((usize)trap);

  // in den User Mode wechseln
  asm volatile("mret");
}
```

Kontextwechsel: User ⟶ Kernel

- sobald ein Kontextwechsel erforderlich wird (z. B. durch *System Call* oder *Interrupt*), muss der **Zustand des Prozesses** zwischengespeichert werden
- danach wird die Kontrolle an den *Trap Handler* des Betriebssystems übertragen

TRAP.S, TEIL 1

```
.globl trap
.globl trap_handler
.align 4

trap:
  # Platz für Register schaffen
  addi sp, sp, -128

  # Register zwischenspeichern
  sw ra, 0(sp)
  sw sp, 4(sp)
  # …

  # Trap Handler mit Parameter aufrufen
  mv a0, sp
  call trap_handler
```

Trap Handler

- sobald ein Kontextwechsel erforderlich wird (z. B. durch *System Call* oder *Interrupt*), muss der **Zustand des Prozesses** zwischengespeichert werden
- danach wird die Kontrolle an den *Trap Handler* des Betriebssystems übertragen

TRAP.C

```
void trap_handler(Stackframe *s) {
  usize pc = r_mepc();
  usize mcause = r_mcause();

  usize nr = s->a7;
  usize param = s->a0;

  // Grund für die Trap auslesen
  usize is_async_exception =
    (mcause & (1ull << 31)) != 0;
  // usize is_syscall = …

  if (is_async_exception) // …
  else if (is_syscall) // …
}
```

Kontextwechsel: Kernel → User

- nach Beendigung des *Trap Handlers* kann die Kontrolle wieder an den User-Prozess übergeben werden
- hierzu müssen die Registerwerte wieder vom Stack geladen werden

TRAP.S, TEIL 2

```
# Register wiederherstellen
lw ra, 0(sp)
lw sp, 4(sp)
# …

# Stack abbauen
addi sp, sp, 128

# zurückspringen
mret
```

Ressourcenzuteilung

QUIZFRAGE

Wie viele Prozesse laufen gerade auf eurem Rechner? Und wie viele Kerne hat er?

Knappheit

Problem:
weit weniger Kerne als Prozesse

Lösung:
zeitliches Multiplexing

- Idee aus der Elektrotechnik
- Ressourcennutzung wird zeitlich aufgeteilt in sogenannte **Zeitscheiben** (en. *time slice*, auch *scheduler quantum* oder *time quantum*)

BASH

```
❯ ps -e | wc -l
713 # eins daneben
❯ nproc
12
```

POWERSHELL

```
❯ (Get-Process).Count
212
❯ [Environment]::ProcessorCount
8
```

Zuteilungsstrategien

en. *Scheduling Strategies*

AUFGABE 3

Welcher Zielkonflikt spielt bei der Auswahl
einer geeigneten Zuteilungsstrategie eine Rolle?

- einerseits: wenige **Unterbrechungen** (en. *preemption*) erwünscht
- andererseits: **Latenz** soll minimiert werden
- nur: Latenzminimierung erfordert Unterbrechungen

Scheduling im Linux-Kernel

VERSIONEN 2.6.33 BIS 6.5

Completely Fair Scheduler

- Prozesse haben ein Gewicht (*weight*) und eine virtuelle Laufzeit (*vruntime*)
- Prozessparameter *load* wird aus *weight* und *vruntime* bestimmt
- anhand von *load* werden die Prozesse auf die Kerne verteilt
- im Hintergrund: sortierte Prozesswarteschlange (Rot-Schwarz-Baum)

Problem: Fairness ist nur eine der Anforderungen, auch Caching, CPU-Auslastung, Energieverbrauch und Latenzanforderungen sind relevant

SEIT VERSION 6.6

Earliest Eligible Virtual Deadline First

- basiert auf *lag* (Differenz aus zugeteilter und tatsächlich erhaltener CPU-Zeit)
- Zeitscheibenlänge ist dynamisch

Ausblick zum Scheduling

- Mehr- und Vielkernarchitekturen
  - Work-Stealing Scheduling
- Energieaspekte
  - Big/Little-Architekturen
  - Linux' "Utilization Clamping" (UCLAMP)
- Dynamische und modulare Ablaufplanung im Betriebssystem
  - Linux sched_ext: Scheduling mittels eBPF-Programmen

Literatur

J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, und A. Fedorova (2016): **„The Linux Scheduler: A Decade of Wasted Cores“**.

I. Stoica und H. Abdel-Wahab (1996): **„Earliest Eligible Virtual Deadline First: A Flexible and Accurate Mechanism for Proportional Share Resource Allocation.“**

Input/Output

REVISITED

Verteilte Zahlen einsammeln

Entwurf in Pseudocode

AUFGABE 2A

Betrachten Sie als gegeben:

> Dateien `1.txt`, `2.txt` … `10000.txt` mit je exakt einer positiven Ganzzahl

Gesucht ist die Summe der Zahlen in allen Textdateien. Formulieren Sie eine passende Lösung in Pseudocode.

Verteilte Zahlen einsammeln

Entwurf in Pseudocode

AUFGABE 2A

Betrachten Sie als gegeben:

> Dateien `1.txt`, `2.txt` … `10000.txt` mit je exakt einer positiven Ganzzahl

Gesucht ist die Summe der Zahlen in allen Textdateien. Formulieren Sie eine passende Lösung in Pseudocode.

```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

Verteilte Zahlen einsammeln

Naive Umsetzung

AUFGABE 2B

Implementieren Sie den Algorithmus als C-Programm.

```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

Verteilte Zahlen einsammeln

Naive Umsetzung

AUFGABE 2B

Implementieren Sie den Algorithmus als C-Programm.

AUFGABE 2C

Welche realen Anwendungen können sich hinter diesem *Toy Example* verbergen?

```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

Fragen?

Literatur

[1] T. Chung und J. Bagur, „The Arduino Guide to Low Power Design“. Zugegriffen: 30. Mai 2025. [Online]. Verfügbar unter: **https://docs.arduino.cc/learn/electronics/low-power/**

[2] J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, und A. Fedorova, „The Linux Scheduler: A Decade of Wasted Cores“, in *Proceedings of the Eleventh European Conference on Computer Systems*, in EuroSys '16. London, United Kingdom: Association for Computing Machinery, 2016. doi: **10.1145/2901318.2901326**.

[3] I. Stoica und H. Abdel-Wahab, „Earliest Eligible Virtual Deadline First: A Flexible and Accurate Mechanism for Proportional Share Resource Allocation“, Northfolk, Virginia, USA, technical report TR-96–22, Jan. 1996. [Online]. Verfügbar unter: **https://people.eecs.berkeley.edu/~istoica/papers/eevdf-tr-95.pdf**

Lizenzhinweise

Kursmaterialien · **Quelltext**

Diese Präsentation wurde zuletzt am 09.06.2026 bearbeitet. Sie basiert auf den Folien der zugehörigen Vorlesung von Prof. Dr. Michael Engel. Sofern nicht anders angegeben, sind die Inhalte unter der **Lizenz CC BY-SA 4.0** verfügbar. Das Universitätslogo sowie die Schriftart UB Scala Sans sind Eigentum der Otto-Friedrich-Universität Bamberg.

Folgende **Open-Source-Komponenten** kommen in der Präsentation zum Einsatz: **circuiteria** (Apache 2.0), **finite** (MIT), **fontawesome** (SIL), **pgf-tikz** (GPL 2.0), **typst** (Apache 2.0), **typst-ccicons** (MIT), **typst-polylux** (MIT).