Betriebssystem-Entwicklung

6. Scheduling

Michael Schöttner

Vorschau

Heinrich Heine
Universität Düsseldorf

- Motivation & Begriffe
- Thread-Umschaltung
- Scheduling
- Scheduler in hhuTOS
- Zusammenfassung

Motivation

Heinrich Heine
Universität Düsseldorf

- Ziel: Anwendungen als eigenständige Threads ausführen
  - Wir möchten einen dynamischen, keinen statischer Ablaufplan für Threads
  - Threads geben die CPU freiwillig ab oder bekommen diese entzogen

- Lösung: **Scheduler** im Betriebssystem
  - Entscheidet welcher Thread als nächstes die CPU bekommt
  - Falls ein Thread blockiert wird durch den Scheduler automatisch auf einen anderen Thread umgeschaltet

Begriffe

Heinrich Heine
Universität Düsseldorf

- **Kooperatives Multithreading**:
  Threads müssen regelmäßig die CPU freiwillig abgeben, da das Betriebssystem die CPU nicht entziehen kann.

- **Präemptives Multithreading**:
  Threads können die CPU freiwillig abgeben,
  bei Bedarf kann diese aber durch das Betriebssystem jederzeit entzogen werden.

Begriffe

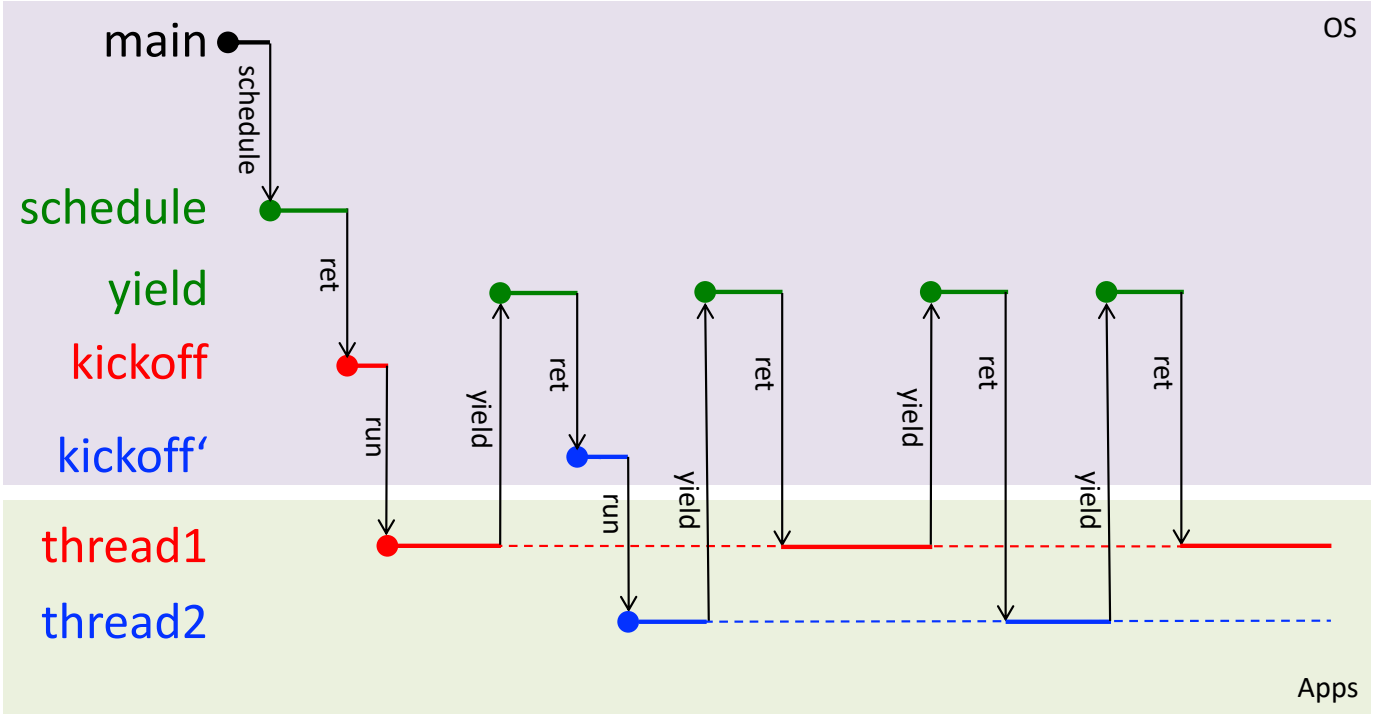
- **Adressraum**: Der zu einem Zeitpunkt direkt zugreifbarer Speicher
- **Prozess**: Adressraum, Thread(s) und Verwaltungsinformationen (Programm, das sich in Ausführung befindet, und seine Daten)
- **Thread**: selbständige Aktivität
  - gehört immer zu einem Prozess
    → teilt sich Daten, Code & Betriebsmittel mit Threads des gleichen Prozesses
  - separater Registersatz & eigener Laufzeitkeller
  - (Manchmal auch als lightweight process bezeichnet)

Vorschau

Heinrich Heine
Universität Düsseldorf

- Motivation & Begriffe
- Thread-Umschaltung
- Scheduling
- Scheduler in hhuTOS
- Zusammenfassung

Kooperative Thread-Umschaltung



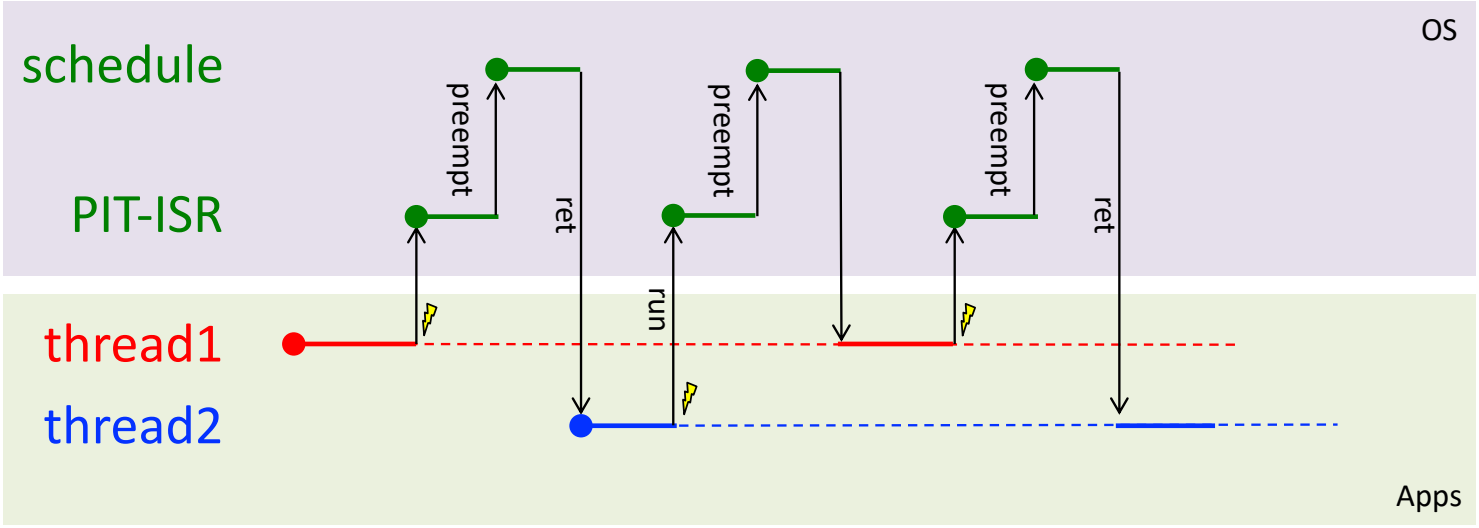
Präemptive Thread-Umschaltung

Heinrich Heine
Universität Düsseldorf

- Problem: Wie kann das BS einem Thread die CPU entziehen?

- Lösung: mithilfe der Zeitgeber-Unterbrechung (engl. timer interrupt)
  - Zeitgeber löst periodisch einen Interrupt aus, z.B. jede jede Millisekunde
  - Das Zeitintervall ist programmierbar
  - Jeder Interrupt ist ein „tick“

Präemptive Thread-Umschaltung



Vorschau

Heinrich Heine
Universität Düsseldorf

- Motivation & Begriffe
- Thread-Umschaltung
- Scheduling
- Scheduler in hhuTOS
- Zusammenfassung

Scheduler

Heinrich Heine
Universität Düsseldorf

- In der Regel warten viele Threads auf die CPU
- Die Entscheidung wer als nächstes die CPU bekommt muss schnell geschehen
- Der Scheduler verwaltet alle rechenbereiten Threads in einer Bereit-Liste (engl. ready queue)
  - In der Praxis werden oft mehrere Listen (für verschiedene Prioritäten) oder Bäume verwendet
- Blockierte Threads werden nicht in der Bereit-Liste gespeichert, da diese sonst jedes Mal übersprungen werden müssten

Scheduling Ziele

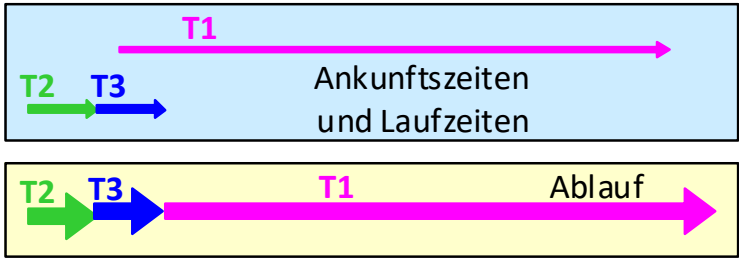
Heinrich Heine
Universität Düsseldorf

- Fairness: Jeder Thread sollte im Mittel den gleichen CPU-Zeitanteil erhalten
- Wartezeit: Wartezeit in der Bereit-Liste minimieren
- Durchsatz: #Threads pro Zeiteinheit sollte maximal sein
- Ausführungszeit:
  - Die Zeitspanne vom Thread-Beginn bis zum Thread-Ende sollte sie minimal sein.
  - Sie enthält alle Zeiten in der Bereit-Liste, die Ausführungs- und Blockiertzeiten
- Antwortzeit:
  - Die Zeit zwischen einer Eingabe und der Übergabe der Antwortdaten an das Ausgabegerät sollte minimal sein (interaktive Systeme)

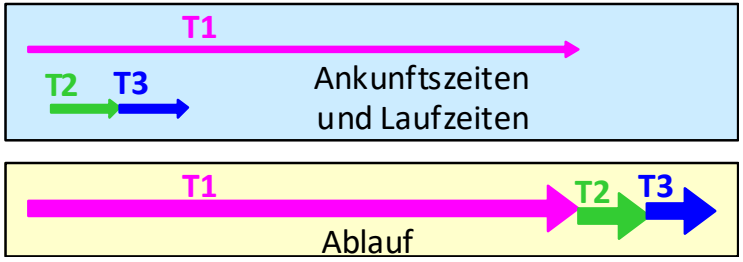
- Problem: teilweise konkurrierende Ziele & Overhead beim Umschalten.

First Come First Served (FCFS)

- Bearbeitung der Threads in Reihenfolge ihrer Ankunft in der Bereitliste.
- Prozessorbesitz bis zum Ende oder freiwilligen Abgabe.
- Beispiel: günstiger Ablauf



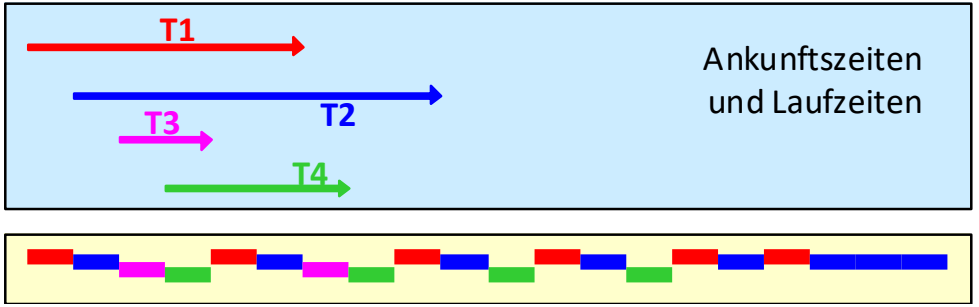
- Beispiel: ungünstiger Ablauf



- Nachteile:
  - Konvoi-Effekt = kurze Threads stauen sich hinter einem langsamen Thread.

Round Robin (RR)

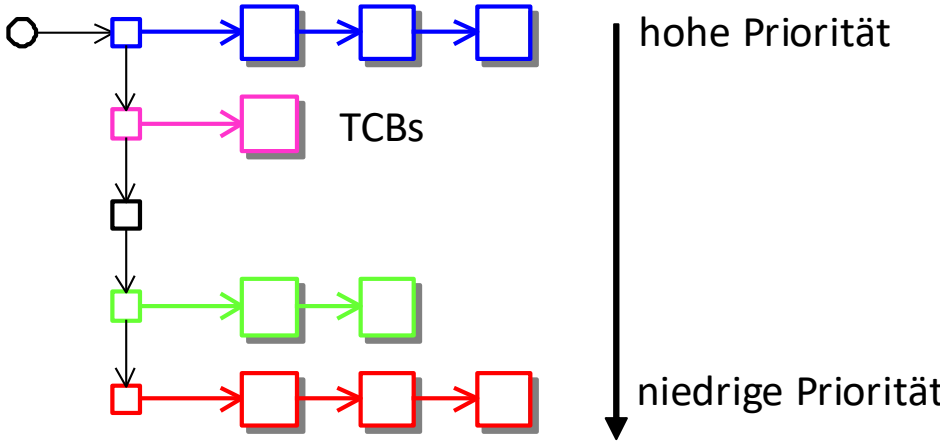
- Ziel: gleichmäßige Verteilung der CPU
- Weit verbreitete Strategie:
  - Threads in Ankunftsreihenfolge verarbeiten
  - Nach Ablauf einer vorher festgesetzten Zeitscheibe (z.B. 10-100ms) findet Verdrängung (engl. preemption) statt und ein Threads wird am Ende der Bereit-Queue eingereiht



- Problem: richtige Wahl der Zeitscheibe / Zeit-Quantum:
  - Vernünftiges Verhältnis zw. Zeit-Quantum und Zeit für den Kontextwechselzeit notwendig
  - Große Zeitscheiben sparen Kontextwechsel, verursachen aber lange Antwortzeiten

Scheduling mit Prioritäten

- Es ist wünschenswert Threads nach Wichtigkeit zu unterscheiden.
  → Jeder Thread erhält eine Prioritätsnummer.
- Scheduler selektiert immer zuerst Thread mit der höchsten Priorität
- Implementierung:
  - Eine Bereit-Liste pro Priorität
  - So kann der richtige Thread schnell gefunden werden
  - TCB = Thread Control Block
    - Infos über den Thread: ID, Stack, etc.



Feedback-Scheduling

Heinrich Heine
Universität Düsseldorf

- Feedback → dynamische Anpassung der Scheduling-Kriterien abhängig vom Verhalten eines Threads (Rechenintensität bzw. Wartezeit)
- Angepasst werden kann die Priorität, Einsortierung in die Bereit-Liste (am Anfang oder am Ende), sowie die Zeitscheibenlänge
- Zwei grundlegende Strategien:
  - wartende Threads hochstufen
  - rechenintensive Threads herabstufen

Leerlauf (engl. idle)

Heinrich Heine
Universität Düsseldorf

- Falls alle Threads warten ist CPU frei u. es läuft ein Leerlauf-Thread:
  - darf nicht anhalten, hat geringste Priorität, muss jederzeit verdrängbar sein.
  - Busy-Looping mit speziellen Instruktionen vermeidbar
    → Beispiel: „HLT“ (x86): stoppt CPU; (Timer-)Interrupt weckt CPU wieder auf.

- Aber Leerlauf auch nutzbar für:
  - Dateisystem defragmentieren
  - Garbage Collection
  - Software-Updates

Multicore Aspekte

Heinrich Heine
Universität Düsseldorf

- Wahllose Kern-Zuteilung schlecht:
  - Hat Thread *t* zuletzt Kern *p* genutzt, so besteht die Chance dass noch Teile seines Adressraums im Cache/TLB von *p* vorhanden sind
  - (Falls ein anderer Thread des gleichen Adressraums zuletzt auf diesem Kern aktiv war)
- Lastausgleich zwischen Kernen durch den Scheduler
- Kern-Affinität durch Programmierer steuerbar → Nachteil: Aufweichung von Prioritäten:
  - evt. zuletzt genutzter Kern belegt und anderer Kern frei
  - falls auf „alten“ Kern gewartet wird,
  - so läuft evt. ein Thread mit niedriger Priorität zuerst

Vorschau

Heinrich Heine
Universität Düsseldorf

- Motivation & Begriffe
- Thread-Umschaltung
- Scheduling
- Scheduler in hhuTOS
- Zusammenfassung

Kooperatives Scheduling in hhuTOS

Heinrich Heine
Universität Düsseldorf

- Kooperativer FCFS-Scheduler
- Eine Bereit-Liste ohne Prioritäten
- Scheduler nimmt immer den ersten Thread aus der Bereit-Liste
- Bei CPU-Abgabe oder Thread-Erzeugung wird immer am Ende der Bereit-Liste eingefügt

Kooperatives Scheduling in hhuTOSc

Heinrich Heine
Universität Düsseldorf

```
class Scheduler : public Dispatcher {
   private:
      Queue readyQueue;                  // auf die CPU wartende Threads

   public:
      Scheduler ();
      void schedule ();                  // Scheduler starten
      void ready (Thread& that);         // Thread in readyQueue eintragen
      void exit ();                      // Thread terminiert sich selbst
      void kill (Thread& that);          // Thread mit 'Gewalt' terminieren
      void yield ();                     // CPU freiwillig abgeben
};
```

Kooperatives Scheduling in hhuTOSr

Heinrich Heine
Universität Düsseldorf

```
pub struct Scheduler {
    active: *mut thread::Thread,
    ready_queue: queue::Queue<Box<thread::Thread>>,
    next_thread_id: u64,
    initialized: bool,
}

impl Scheduler {
  pub fn schedule ();                                  // Scheduler starten

  // Thread in readyQueue eintragen
  pub fn ready (that: Box<dyn thread::ThreadEntry>) -> u64;

  pub fn exit (that: *mut thread::Thread);             // Thread terminiert sich selbst
  pub fn kill (tokill_tid: u64);                       // Thread mit 'Gewalt' terminieren
  pub fn yield_cpu (that: *mut thread::Thread);        // CPU freiwillig abgeben
}
```

Vorschau

Heinrich Heine
Universität Düsseldorf

- Motivation & Begriffe
- Thread-Umschaltung
- Scheduling
- Scheduler in hhuTOS
- Zusammenfassung

Zusammenfassung

Heinrich Heine
Universität Düsseldorf

- Thread ermöglichen Nebenläufigkeit in Anwendungen
- Wenn ein Thread blockiert ist, so kann ein anderer Thread rechnen
- Welcher Thread als nächstes die CPU bekommt entscheidet der Scheduler
- Beim präemptiven Multithreading wird die CPU mithilfe des Timer-Interrupts entzogen