

Probleme

- Problematisch an der Implementierung des Beispiels auf der letzten Folie ist, dass der Zeiger auf den Auftrag `job` nicht geschützt ist und es so zu fehlerhaften Ausführungen kommen kann.
 - Durch Verwendung eines zusätzlichen Semaphors kann dieses Problem behoben werden.
 - Ist die Zeit zwischen zwei Aufträgen zu kurz um die rechtzeitige Bearbeitung sicherzustellen, so kann es zu weiteren Problemen kommen:
 - Problem 1: Der Prozess **Contractor** muss warten, weil der Prozeß **Worker** den letzten Auftrag noch bearbeitet.
 - Problem 2: Der letzte Auftrag wird überschrieben, falls dieser noch gar nicht bearbeitet wurde. Abhängig von der Implementierung des Semaphors könnte dann der neue Auftrag zudem zweifach ausgeführt werden.
- mit Semaphoren sind nur einfache Signalisierungsprobleme (ohne Datentransfer) zu lösen, ansonsten sollten Warteschlangen verwendet werden

Signalisierung durch Semaphore: Leser-Schreiber-Beispiel

- Vorherige Lösung:

Reader:

```
...  
  
    down(semWriter);  
    down(semCounter);  
    rcounter++;  
    up(semCounter);  
    up(semWriter);  
  
    read();  
  
    down(semCounter);  
    rcounter--;  
    up(semCounter);  
  
...
```

Writer:

```
...  
  
    down(semWriter);  
  
    while(true) ← Problem: Busy Waiting  
    {  
        down(semCounter);  
        if(rcounter==0)  
            break;  
        up(semCounter);  
    }  
  
    up(semCounter);  
  
    write();  
  
    up(semWriter);  
  
...
```

Signalisierung durch Semaphore: Leser-Schreiber-Beispiel

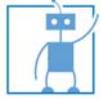
- Lösung mit Signalisierung:

Reader:

```
...  
    down(semWriter);  
    down(semCounter);  
    rcounter++;  
    if(rcounter==1)  
        down(semReader);  
    up(semCounter);  
    up(semWriter);  
  
    read();  
  
    down(semCounter);  
    rcounter--;  
    if(rcounter==0)  
        up(semReader);  
    up(semCounter);  
...
```

Writer:

```
...  
    down(semWriter);  
    down(semReader);  
    up(semReader);  
  
    write();  
  
    up(semWriter);  
...
```

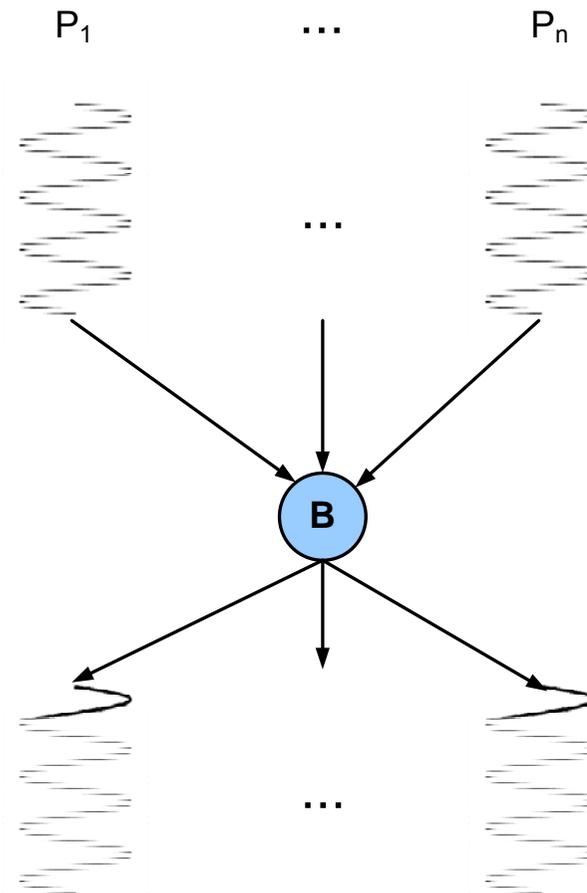


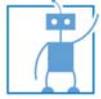
Nebenläufigkeit

Synchrone Kommunikation: Barrieren

Synchrone Kommunikation: Barrieren

- **Definition:** Eine Barriere für eine Menge M von Prozessen ist ein Punkt, den alle Prozesse $P_i \in M$ erreichen müssen, bevor irgendein Prozess aus M die Berechnung über diesen Punkt hinaus fortfahren kann.
- Der Spezialfall für $|M|=2$ wird als Rendezvous, siehe auch Ada, bezeichnet.
- Barrieren können mit Hilfe von Semaphoren implementiert werden.





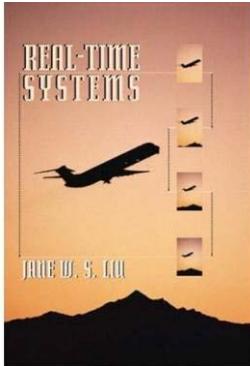
Kapitel 5

Scheduling

Inhalt

- Definitionen
- Kriterien zur Auswahl des Scheduling-Verfahrens
- Scheduling-Verfahren
- Prioritätsinversion
- Exkurs: Worst Case Execution Times

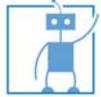
Literatur



Jane W. S. Liu, Real-Time
Systems, 2000

Fridolin Hofmann: Betriebssysteme -
Grundkonzepte und Modellvorstellungen, 1991

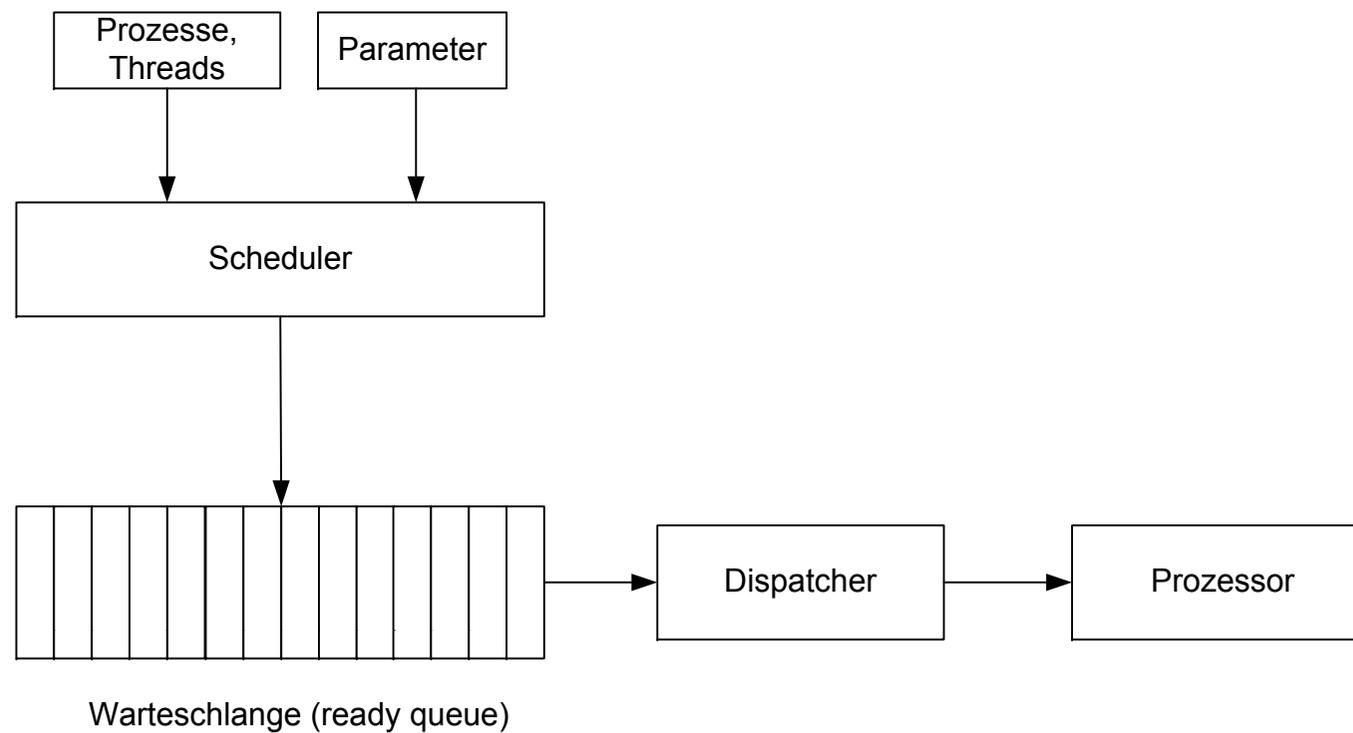
- Journals:
 - John A. Stankovic, Marco Spuri, Marco Di Natale, and Giorgio C. Buttazzo: Implications of classical scheduling results for real-time systems. IEEE Computer, Special Issue on Scheduling and Real-Time Systems, 28(6):16–25, June 2005.
 - Giorgio C. Buttazzo: Rate Monotonic vs. EDF: Judgement Day (<http://www.cas.mcmaster.ca/~downnd/rtsj05-rmedf.pdf>)
 - Puschner, Peter; Burns, Alan: A review of Worst-Case Execution-Time Analysis, Journal of Real-Time Systems 18 (2000), S.115-128



Scheduling

Definitionen

Scheduler und Dispatcher

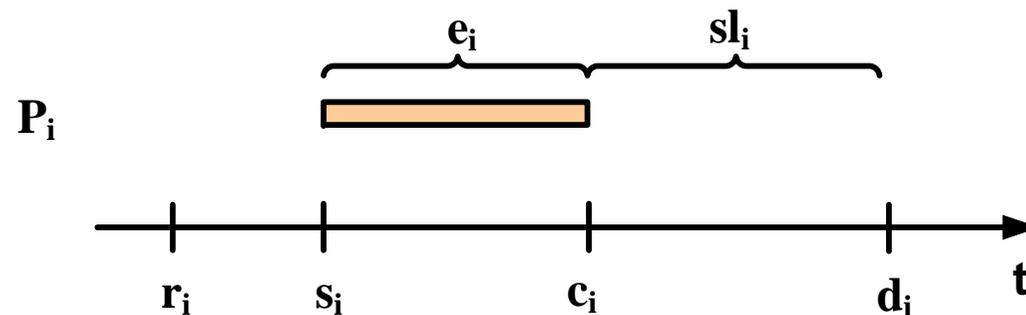


Scheduler und Dispatcher

- **Scheduler:** Modul eines Betriebssystems, das die Rechenzeit an die unterschiedlichen Prozesse verteilt. Der ausgeführte Algorithmus wird als Scheduling-Algorithmus bezeichnet. Aufgabe des Schedulers ist also die langfristige Planung (Vergleich: Erstellung eines Zugfahrplans).
- **Dispatcher:** Übersetzung: Einsatzleiter, Koordinator, Zuteiler (v.a. im Bereich der Bahn gebräuchlich). Im Rahmen der Prozessverwaltung eines Betriebssystems dient der Dispatcher dazu, bei einem Prozesswechsel dem derzeit aktiven Prozess die CPU zu entziehen und anschließend dem nächsten Prozess die CPU zuzuteilen. Die Entscheidung, welcher Prozess der nächste ist, wird vom Scheduler im Rahmen der Warteschlangenorganisation getroffen.

Zeitliche Bedingungen

- Folgende Größen sind charakteristisch für die Ausführung von Prozessen:
 - P_i bezeichnet den i. **Prozess** (bzw. Thread)
 - r_i : **Bereitzeit (ready time)** des Prozesses P_i und damit der früheste Zeitpunkt an dem der Prozess dem Prozessor zugeteilt werden kann.
 - s_i : **Startzeit**: der Prozessor beginnt P_i auszuführen.
 - e_i : **Ausführungszeit (execution time)**: Zeit die der Prozess P_i zur reinen Ausführung auf dem Prozessor benötigt.
 - c_i : **Abschlußzeit (completion time)**: Zeitpunkt zu dem die Ausführung des Prozesses P_i beendet wird.
 - d_i : **Frist (deadline)**: Zeitpunkt zu dem die Ausführung des Prozesses P_i in jeden Fall beendet sein muss.
 - sl_i : **Slack**: Deadline-(aktuelle Zeit + verbleibende Berechnungszeit)



Spielraum (slack time)

- Mit dem Spielraum (slack time) sl_i eines Prozesses P_i wird Zeitraum bezeichnet, um den ein Prozess noch maximal verzögert werden darf:
 - Die Differenz zwischen der verbleibenden Zeit bis zum Ablauf der Frist und der noch benötigten Ausführungszeit zur Beendigung des Prozesses P_i .
- Der Spielraum eines Prozesses, der aktuell durch den Prozessor ausgeführt wird, bleibt konstant, während sich die Spielräume aller nicht ausgeführten Prozesse verringern.

Faktoren bei der Planung

- Für die Planung des Scheduling müssen folgende Faktoren berücksichtigt werden:
 - Art der Prozesse (periodisch, nicht periodisch, sporadisch)
 - Gemeinsame Nutzung von Ressourcen (**shared resources**)
 - Fristen
 - Vorrangrelationen (**precedence constraints**: Prozess P_i muss vor P_j ausgeführt werden)

Arten der Planung

- Es kann zwischen unterschiedlichen Arten zum Planen unterschieden werden:
 - offline vs. online Planung
 - statische vs. dynamische Planung
 - präemptives vs. nicht-präemptives Scheduling

Offline Planung

- Mit der offline Planung wird die Erstellung eines Ausführungsplanes zur Übersetzungszeit bezeichnet. Zur Ausführungszeit arbeitet der Dispatcher den Ausführungsplan dann ab.
- **Vorteile:**
 - deterministisches Verhalten des Systems
 - wechselseitiger Ausschluss in kritischen Bereichen wird direkt im Scheduling realisiert
- **Nachteile:**
 - Bereitzeiten, Ausführungszeiten und Abhängigkeit der einzelnen Prozesse müssen schon im Voraus bekannt sein.
 - Die Suche nach einem Ausführungsplan ist im Allgemeinen ein NP-hartes Problem. Es werden jedoch keine optimalen Pläne gesucht, vielmehr ist eine gute Lösung (Einhaltung aller Fristen) ausreichend.

Online Scheduling

- Alle Schedulingentscheidungen werden online, d.h. auf der Basis der Menge der aktuell lauffähigen Prozesse und ihrer Parameter getroffen.
- Im Gegensatz zur offline Planung muss wechselseitiger Ausschluss nun über den expliziten Ausschluss (z.B. Semaphoren) erfolgen.
- Vorteile:
 - Flexibilität
 - Bessere Auslastung der Ressourcen
- Nachteile:
 - Es müssen zur Laufzeit Berechnungen zum Scheduling durchgeführt werden ⇒ Rechenzeit geht verloren.
 - Garantien zur Einhaltung von Fristen sind schwieriger zu geben.
 - Problematik von Race Conditions

Statische vs. dynamische Planung

- Bei der statischen Planung basieren alle Entscheidungen auf Parametern, die vor der Laufzeit festgelegt werden.
- Zur statischen Planung wird Wissen über:
 - die Prozessmenge
 - ihre Prioritäten
 - das Ausführungsverhalten

benötigt.

- Bei der dynamischen Planung können sich die Scheduling-Parameter (z.B. die Prioritäten) zur Laufzeit ändern.
- **Wichtig:** Statische Planung und Online-Planung schließen sich nicht aus: z.B. Scheduling mit festen Prioritäten.

Präemption

- Präemptives (bevorrechtigt, entziehend) Scheduling: Bei jedem Auftreten eines relevanten Ereignisses wird die aktuelle Ausführung eines Prozesses unterbrochen und eine neue Schedulingentscheidung getroffen.
- Präemptives (unterbrechbares) Abarbeiten:
 - Aktionen (Prozesse) werden nach bestimmten Kriterien geordnet (z.B. Prioritäten, Frist,...).
 - Diese Kriterien sind statisch festgelegt oder werden dynamisch berechnet.
 - Ausführung einer Aktion wird sofort unterbrochen, sobald Aktion mit höherer Priorität eintrifft.
 - Die unterbrochene Aktion wird an der Unterbrechungsstelle fortgesetzt, sobald keine Aktion höherer Priorität ansteht.
 - Typisch für Echtzeitaufgaben (mit Ausnahme von Programmteilen, die zur Sicherung der Datenkonsistenz nicht unterbrochen werden dürfen).
 - Nachteil: häufiges Umschalten reduziert Leistung.

Ununterbrechbares Scheduling

- Ein Prozess, der den Prozessor zugewiesen bekommt, wird solange ausgeführt, bis der Prozess beendet wird oder er aber den Prozess freigibt.
- Scheduling-Entscheidungen werden nur nach der Prozessbeendigung oder dem Übergang des ausgeführten Prozesses in den blockierten Zustand vorgenommen.
- Eine begonnene Aktion wird beendet, selbst wenn während der Ausführung Aktionen höherer Dringlichkeit eintreffen
⇒ Nachteil: evtl. Versagen (zu lange Reaktionszeit) des Systems beim Eintreffen unvorhergesehener Anforderungen
- Anmerkung: Betriebssysteme unterstützen allgemein präemptives Scheduling solange ein Prozess im Userspace ausgeführt, Kernelprozesse werden häufig nicht oder selten unterbrochen.
⇒ Echtzeitbetriebssysteme zeichnen sich in Bezug auf das Scheduling dadurch aus, dass nur wenige Prozesse nicht unterbrechbar sind und diese wiederum sehr kurze Berechnungszeiten haben.