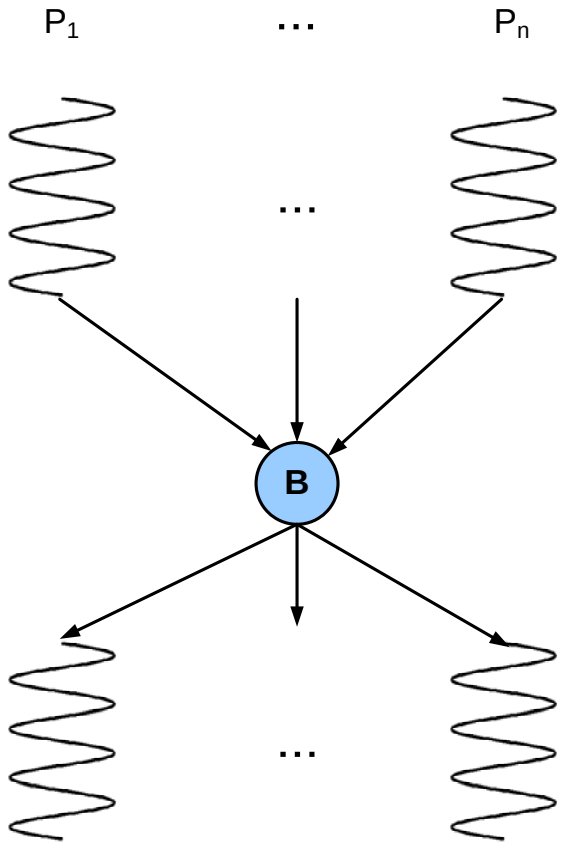
Nebenläufigkeit

Synchrone Kommunikation: Barrieren, Occam

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.



Occam

- Als Programmiersprache wurde Occam verwendet, mit der man parallel Abläufe festlegen konnte.
- Als Namenspate fungierte der Philosoph William of Ockham. Sein Postulat „*Dinge sollten nicht komplizierter als unbedingt notwendig gemacht werden*“ war Motto der Entwicklung.
- Occam basiert auf dem Modell CSP (communicating sequential processes) von C.A.R. Hoare; siehe auch CCS (Calculus of Communicating Systems) von R. Milner
- Occam ist eine Sprache, die die parallele Ausführung von Aktionen direkt mit einbezieht
- Die Kommunikation zwischen den einzelnen Prozessen erfolgt synchron über unidirektionale Kanäle.
- Die Realisierung auf dem Transputer ist 1:1. Als Kanal zwischen zwei Prozessen auf unterschiedlichen Transputern kann ein (halber) Link benutzt werden. Befinden sich die beiden Prozesse auf einem Transputer, so kann der Kanal über Speicherplätze simuliert werden.
- Siehe http://vl.fmnet.info/occam/

William of Ockham

C.A.R. Hoare

Occam

- Code wird in Occam zu Blöcken zusammengefasst, indem die einzelnen Zeilen alle gleichweit eingerückt werden
- Eine Anweisung wird durch das Ende der Zeile beendet
- Sprachelemente:
  - Eingabe ? :
  - Ausgabe !:
  - Sequentielle Ausführung SEQ:
  - Parallele Ausführung PAR:
  - Alternative Ausführung ALT*:

```
keyboard ? c
```

```
screen ! c
```

```
SEQ
    x:=1
    y:=2
```

```
PAR
    keyboard ? x
    screen ! y
```

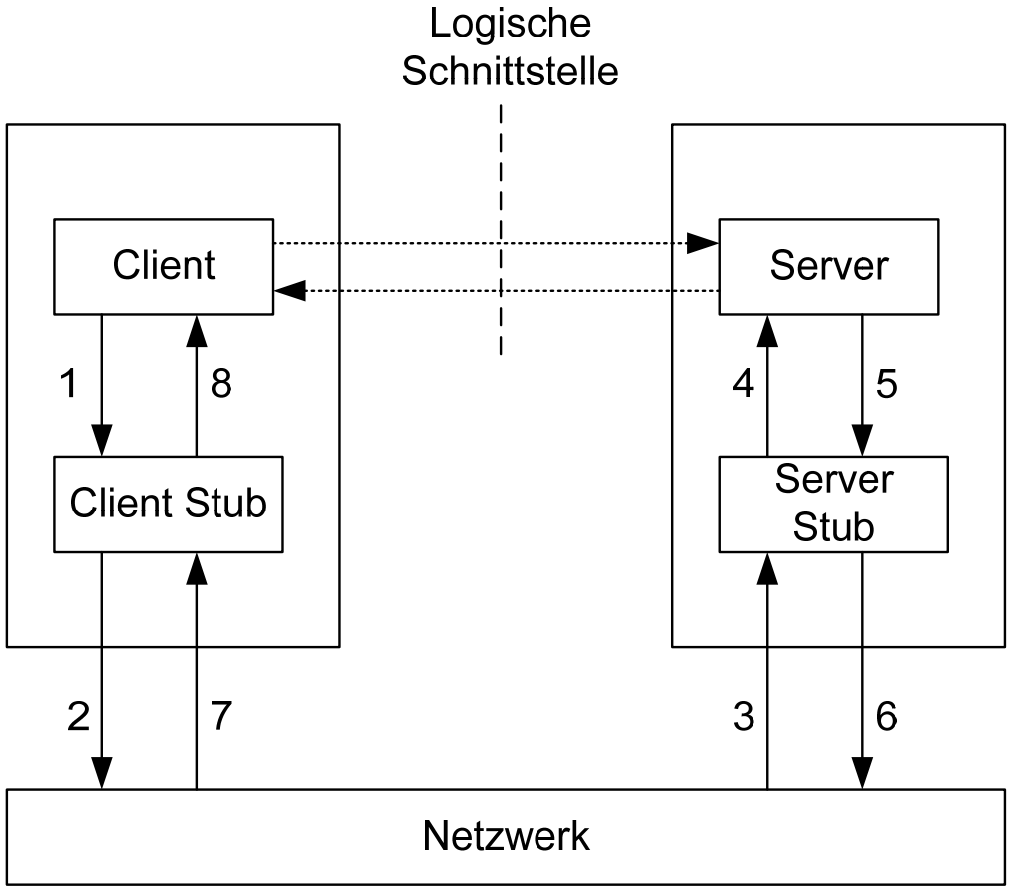
```
ALT
    x<10 & chan1 ? y
        screen ! y
    x<20 & chan2 ? y
        screen ! y
```

*Bei der ALT kann für jeden Block eine Bedingung, sowie eine Eingabe (beide optional) angegeben werden. Es wird derjenige Block ausgeführt, dessen Bedingung wahr ist und auf dem Daten eingehen. Trifft dies für mehrere Blöcke zu, so wird ein Block gewählt und ausgeführt.

Nebenläufigkeit

Funktionsaufrufe als Kommunikation

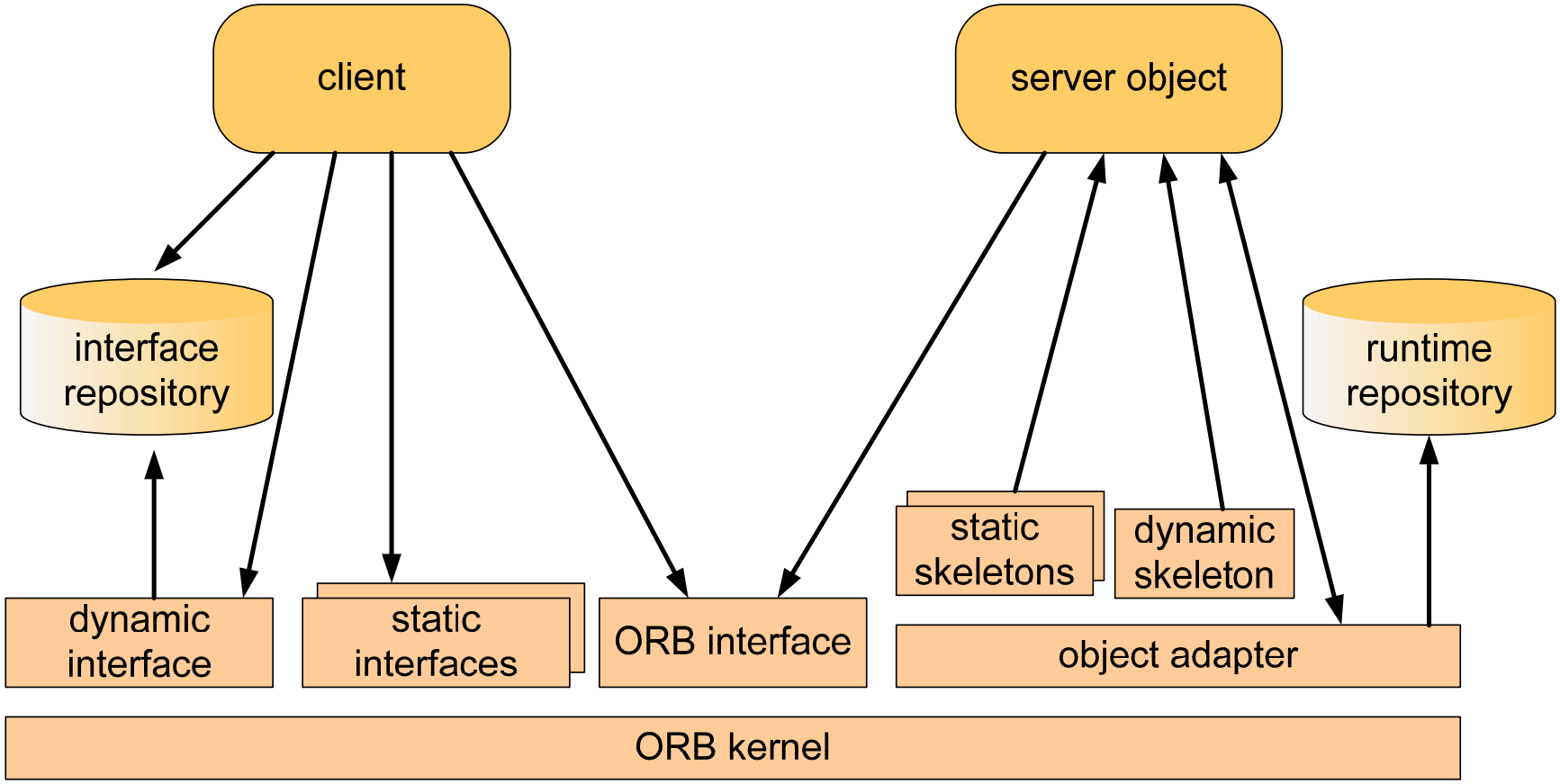
Remote Procedure Call (RPC)



Ablauf RPC

- Bei einem Funktionsaufruf über RPC werden folgende Schritte ausgeführt:
  1. Lokaler Funktionsaufruf vom Client an Client Stub
  2. Konvertierung des Funktionsaufrufs in Übertragungsformat und Senden der Nachricht
  3. Empfang der Nachricht von Kommunikationschicht
  4. Entpacken der Nachricht und lokaler Funktionsaufruf
  5. Übermittlung des Ergebnisses von Server an Server Stub
  6. Konvertierung des Funktionsergebnisses in Übertragungsformat und Senden der Nachricht
  7. Empfang der Nachricht von Kommunikationschicht
  8. Entpacken der Nachricht und Übermittlung des Ergebnisses an Client
- Voraussetzung für Echtzeitfähigkeit: Echtzeitfähiges Kommunikationsprotokoll und Mechanismus zum Umgang mit Nachrichtenverlust

Corba (Common Object Request Broker Architecture)



Komponenten in Corba

- **ORB (Object Request Broker)**: vermittelt Anfragen zwischen Server und Client, managt die Übertragung, mittlerweile sind auch echtzeitfähige ORBs verfügbar
- **ORB Interface**: Schnittstelle für Systemdienstaufrufe
- **Interface repository**: speichert die Signaturen der zur Verfügung stehenden Schnittstellen, die Schnittstellen werden dabei in der IDL-Notation (Interface Definition Language) gespeichert.
- **Object Adapter**: Überbrückt die Lücke zwischen Corba-Objekten mit IDL-Schnittstelle und Serverobjekten in der jeweiligen Programmiersprache
- **Runtime repository**: enthält die verfügbaren Dienste und die bereit instantiierten Objekte mitsamt den entsprechenden IDs
- **Skeletons**: enthalten die Stubs für die Serverobjektaufrufe

Nebenläufigkeit

Zusammenfassung & Wiederholung

Zusammenfassung

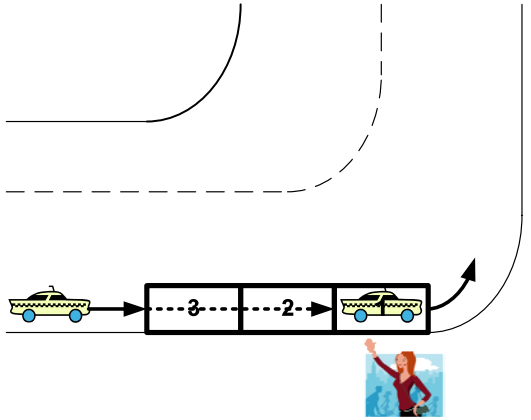
- Folgende Fragen wurden in dieser Vorlesung erklärt und sollten nun verstanden sein:
  - Was ist Nebenläufigkeit / Parallelität?
  - Mit welchen Techniken kann man Nebenläufigkeit erreichen und wann wird welche Technik angewendet?
  - Wie können **race conditions** vermieden werden?
  - Welche Arten der Interprozesskommunikation gibt es (+allgemeine Erklärung)?

Klausurfragen - Nebenläufigkeit

- Klausur WS 06/07
  - Nennen Sie zwei Gründe, wieso nebenläufige Programmierung häufig in Echtzeitsystemen angewandt wird.
  - Was sind Race Conditions?
- Wiederholungsklausur WS 06/07:
  - In der Vorlesung haben Sie die Konzepte Nachrichtenwarteschlangen, Semaphor, sowie Barrieren kennengelernt.
    a) Beschreiben Sie, wie man mit Hilfe von Semaphoren Nachrichtenwarteschlangen implementieren kann.
    b) Beschreiben Sie, wie man mit Hilfe von Nachrichtenwarteschlangen Semaphoren implementieren kann.
    c) Beschreiben Sie, wie man mit Hilfe von Semaphoren Barrieren implementieren kann.
- Klausur WS 07/08
  - **Annahme**: Für folgende Aufgaben können Sie davon ausgehen, dass maximal ein Signal pro Runde an den Automaten geschickt wird. Die Prozesse, die die modellierten Komponenten benutzen, müssen Sie nicht modellieren. Vermeiden Sie Busy Waiting.
  - Modellieren Sie einen Automaten, der das Rendezvouskonzept umsetzt.
  - Modellieren Sie eine Komponente, die das Leser-Schreiber-Problem für 1 Schreiber und beliebig viele Leser mit Schreiberpriorität umsetzt.

Klausur WS07/08 - Nebenläufigkeit

- Gegeben Sie folgendes Szenario: am Münchner Odeonsplatz gibt es eine Wartebucht für Taxis. Zur Vereinfachung gehen wir davon aus, dass die Wartebucht aus drei Plätzen besteht und immer nur ein Passagier gleichzeitig auf ein Taxi wartet. Passagiere steigen an der ersten Wartebucht ein, die Taxis rücken nach, sobald das Taxi vor ihnen losgefahren ist. Implementieren Sie nun schrittweise eine Prozesssynchronisation, so dass es zu keinen Auffahrunfällen kommt, die Taxis in der Ankunftsreihenfolge auch wieder losfahren, Taxis nur mit Passagier losfahren, Passagiere nicht aus Versehen ein nicht-existentes Taxi betreten und es zu keinen Verklemmungen kommt.



a) Notieren Sie die wichtigen Programmabschnitte des Taxiprozesses und des Passagierprozesses. Lassen Sie genügend Platz für spätere Synchronisationsoperationen.
Beispiel: `fahreInErsteWartebucht();`

b) Geben Sie die zur Synchronisation der Taxis und Passagiere benötigten Semaphore, sowie der Initialwerte an. Gehen Sie dabei davon aus, dass zu Beginn kein Taxi in der Wartebucht und keine wartenden Passagiere vorhanden sind.
Beispiel: `semTaxi(1)` würde bedeuten, Sie verwenden einen Semaphor semTaxi, der mit 1 initialisiert ist.
`int i=0;` wenn sie eine ganzzahlige Variable mit Initialisierungswert 1 benutzen wollen.

c) Ergänzen Sie den Taxiprozess und Passagierprozess mit passenden up() und down()-Methoden, um die Aufgabenstellung zu erfüllen.
Beispiel: `down(semTaxi);` bedeutet das Anfordern des Semaphors semTaxi
Beispiel: `up(semTaxi);` bedeutet das Freigeben des Semaphors semTaxi

d) Der Wartebereich am Odeonsplatz ist begrenzt. Stellen Sie sicher, dass maximal 3 Taxis auf Fahrgäste warten und kein Rückstau entsteht. Die Überprüfung ob der Wartebereich belegt ist, soll dabei so schnell wie möglich erfolgen um den Straßenverkehr nicht zu behindern. Andererseits, sollen die Taxifahrer auf jeden Fall in den letzten Warteplatz fahren, falls dieser frei ist.

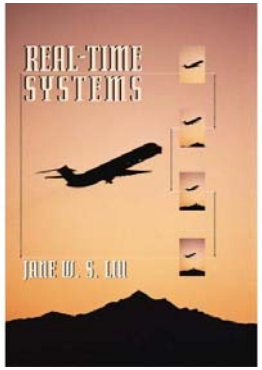
Kapitel 4

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

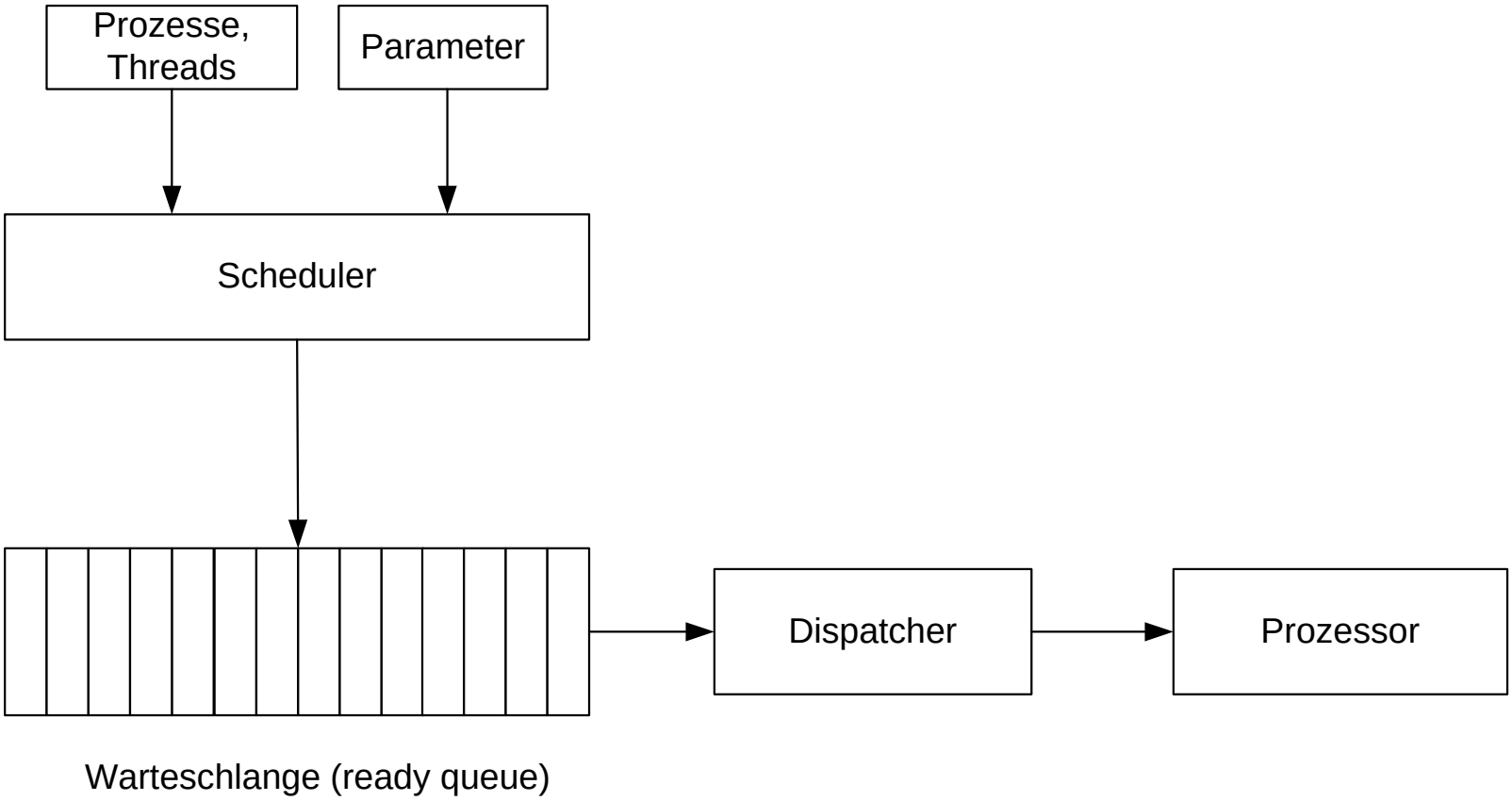
Fridolin Hofmann: Betriebssysteme - Grundkonzepte und Modellvorstellungen, 1991

- 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/~downd/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

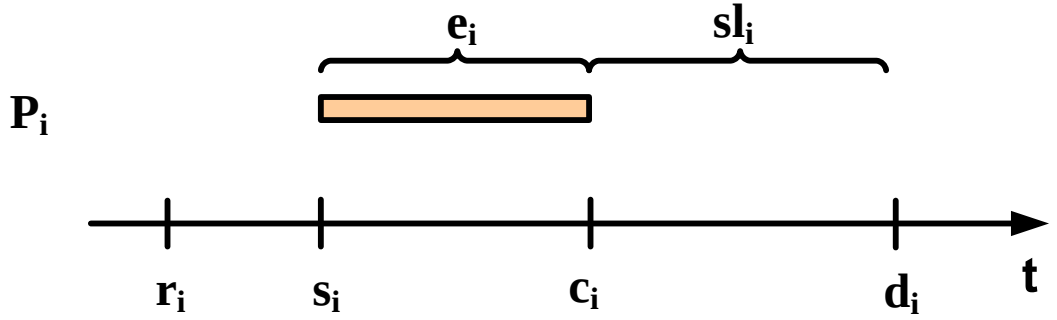


Scheduler und Dispatcher

- **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:
  1. P_i bezeichnet den i. **Prozess** (bzw. Thread)
  2. r_i: **Bereitzeit** (**ready time**) des Prozesses P_i und damit der früheste Zeitpunkt an dem der Prozess dem Prozessor zugeteilt werden kann.
  3. s_i: **Startzeit**: der Prozessor beginnt P_i auszuführen.
  4. e_i: **Ausführungszeit** (**execution time**): Zeit die der Prozess P_i zur reinen Ausführung auf dem Prozessor benötigt.
  5. c_i: **Abschlußzeit** (**completion time**): Zeitpunkt zu dem die Ausführung des Prozesses P_i beendet wird.
  6. d_i: **Frist** (**deadline**): Zeitpunkt zu dem die Ausführung des Prozesses P_i in jeden Fall beendet sein muss.



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 Pi .
- 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 Schedulings müssen folgende Faktoren berücksichtigt werden:
  - Art der Prozesse (periodisch, nicht periodisch, sporadisch)
  - Gemeinsame Nutzung von Ressourcen (**shared ressources**)
  - 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 ein 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 Berechungen zum Scheduling durchgeführt werden $\Rightarrow$ 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.

Schedulingkriterien

- Kriterien in Standardsystemen sind:
  - Fairness: gerechte Verteilung der Prozessorzeit
  - Effizienz: vollständige Auslastung der CPU
  - Antwortzeit: interaktive Prozesse sollen schnell reagieren
  - Verweilzeit: Aufgaben im Batchbetrieb (sequentielle Abarbeitung von Aufträgen) sollen möglichst schnell ein Ergebnis liefern
  - Durchsatz: Maximierung der Anzahl der Aufträge, die innerhalb einer bestimmten Zeitspanne ausgeführt werden
- In Echtzeitsystemen:
  - Einhaltung der Fristen: d.h. $\forall i\ c_i<d_i$ unter Berücksichtigung von Kausalzusammenhängen (Synchronisation, Vorranggraphen, Präzedenzsystemen)
  - Zusätzliche Kriterien können anwendungsabhängig hinzugenommen werden, solange sie der Einhaltung der Fristen untergeordnet sind.

Scheduling

Verfahren

Allgemeines Verfahren

- Gesucht: Plan mit aktueller Start und Endzeit für jeden Prozess P_i.
- Darstellung zum Beispiel als nach der Zeit geordnete Liste von Tupeln (P_i, s_i, c_i)
- Falls Prozesse unterbrochen werden können, so kann jedem Prozess P_i auch eine Menge von Tupeln zugeordnet werden.
- Phasen der Planung:
  - Test auf Einplanbarkeit (feasibility check)
  - Planberechnung (schedule construction)
  - Umsetzung auf Zuteilung im Betriebssystem (dispatching)
- Bei Online-Verfahren können die einzelnen Phasen überlappend zur Laufzeit ausgeführt werden.
- Zum Vergleich von Scheduling-Verfahren können einzelne Szenarien durchgespielt werden.

Definitionen

- **Zulässiger Plan**: Ein Plan ist zulässig, falls alle Prozesse einer Prozessmenge eingeplant sind und dabei keine Präzedenzrestriktionen und keine Zeitanforderungen verletzt werden.
- **Optimales Planungsverfahren**: Ein Verfahren ist optimal, falls es für jede Prozessmenge unter gegebenen Randbedingung einen zulässigen Plan findet, falls ein solcher existiert.

Test auf Einplanbarkeit

- Zum Test auf Einplanbarkeit können zwei Bedingungen angegeben werden, die für die Existenz eines zulässigen Plans notwendig sind (Achtung: häufig nicht ausreichend):
  1. $r_i + e_i \leq d_i$, d.h. jeder Prozess muss in dem Intervall zwischen Bereitzeit und Frist ausgeführt werden können.
  2. Für jeden Zeitraum $[t_i, t_j]$ muss die Summe der Ausführungszeiten e_x der Prozesse P_x mit $r_x \geq t_i \wedge d_x \leq$ tj kleiner als der Zeitraum sein.
- Durch weitere Rahmenbedingungen (z.B. Abhängigkeiten der einzelnen Prozesse) können weitere Bedingungen hinzukommen.

Schedulingverfahren

- Planen aperiodischer Prozesse
  - Planen durch Suchen
  - Planen nach Fristen
  - Planen nach Spielräumen
- Planen periodischer Prozesse
  - Planen nach Fristen
  - Planen nach Raten
- Planen abhängiger Prozesse