

Message Queues in POSIX

- POSIX definiert folgende Funktionen für Nachrichtenwarteschlangen:

```
mqd_t mq_open(const char *name, int oflag, ...); /*Oeffnen einer Message Queue*/
int mq_close(mqd_t mqdes); /*Schliessen einer Message Queue*/
int mq_unlink(const char *name); /*Loeschen einer
    Nachrichtenwarteschlange*/

int mq_send(mqd_t mqdes, const char *msg_ptr,
    size_t msg_len, unsigned int msg_prio); /*Senden einer Nachricht*/
size_t mq_receive(mqd_t mqdes, char *msg_ptr,
    size_t msg_len, unsigned int *msg_prio); /*Empfangen einer Nachricht*/
int mq_setattr(mqd_t mqdes, const struct
    mq_attr *mqstat, struct mq_attr *mqstat); /*Aendern der Attribute*/
int mq_getattr(mqd_t mqdes,
    struct mq_attr *mqstat); /*Abrufen der aktuellen
    Eigenschaften*/

int mq_notify(mqd_t mqdes,
    const struct sigevent *notification); /*Anforderung eines Signals bei
    Nachrichtenankunft*/
```



Nebenläufigkeit

IPC: Kommunikation durch Ereignisse

Signale

- **Signale** werden in Betriebssystemen typischerweise zur Signalisierung von Ereignissen an Prozessen verwendet.
- Signale können verschiedene Ursachen haben:
 - Ausnahmen, z.B. Division durch Null (SIGFPE) oder ein Speicherzugriffsfehler (SIGSEGV)
 - Reaktion auf Benutzereingaben (z.B. Ctrl / C)
 - Signal von anderem Prozess zur Kommunikation
 - Signalisierung von Ereignissen durch das Betriebssystem, z.B. Ablauf einer Uhr, Beendigung einer asynchronen I/O-Funktion, Nachrichtankunft an leerer Nachrichtenwarteschlange (siehe `mq_notify()`)

Prozessreaktionen auf Signale

- Der Prozess hat drei Möglichkeiten auf Signale zu reagieren:
 1. Ignorierung der Signale
 2. Ausführen einer Signalbehandlungsfunktion
 3. Verzögerung des Signals, bis Prozess bereit für Reaktion ist
- Zudem besteht die Möglichkeit mit der Standardreaktion auf das bestimmte Signal zu reagieren. Da aber typischerweise die Reaktion auf Signale die Beendigung des Empfängerprozesses ist, sollte ein Programm über eine vernünftige Signalbehandlung verfügen, sobald ein Auftreten von Signalen wahrscheinlich wird.

Semaphore zur Vermittlung von Ereignissen

- Semaphore können neben der Anwendung des wechselseitigen Ausschlusses auch zur Signalisierung von Ereignissen verwendet werden.
- Es ist zulässig, dass Prozesse (Erzeuger) Semaphore andauernd freigeben und andere Prozesse (Verbraucher) Semaphore dauern konsumieren.
- Es können auch benannte Semaphore erzeugt werden, die dann über Prozessgrenzen hinweg verwendet werden können.
- Notwendige Funktionen sind dann:
 - `sem_open()`: zum Erzeugen und / oder Öffnen eines benannten Semaphors
 - `sem_unlink()`: zum Löschen eines benannten Semaphors

Signalisierung durch Semaphore: Beispiel

- Beispiel: ein Prozeß **Worker** wartet auf einen Auftrag (abgespeichert z.B. in einem char-Array job) durch einen Prozess **Contractor**, bearbeitet diesen und wartet im Anschluß auf den nächsten Auftrag:

Worker*:

```
while(true)
{
    down(sem); /*wait for
               next job*/
    execute(job);
}
```

Contractor*:

```
...
job=... /*create new job and save
        address in global variable*/
up(sem); /*signal new job*/
...
```

** sehr stark vereinfachte Lösung, da zu einem Zeitpunkt nur ein Job verfügbar sein darf*

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);
...
```

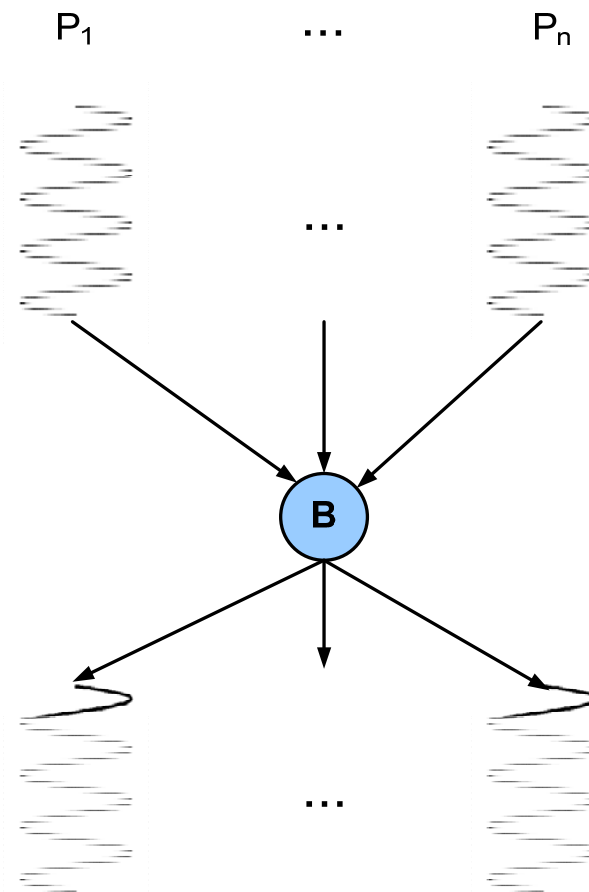


Nebenläufigkeit

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.





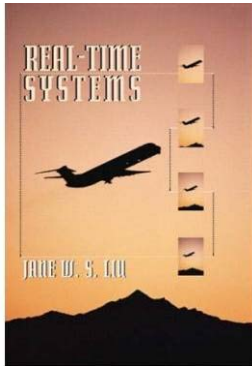
Kapitel 4

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:

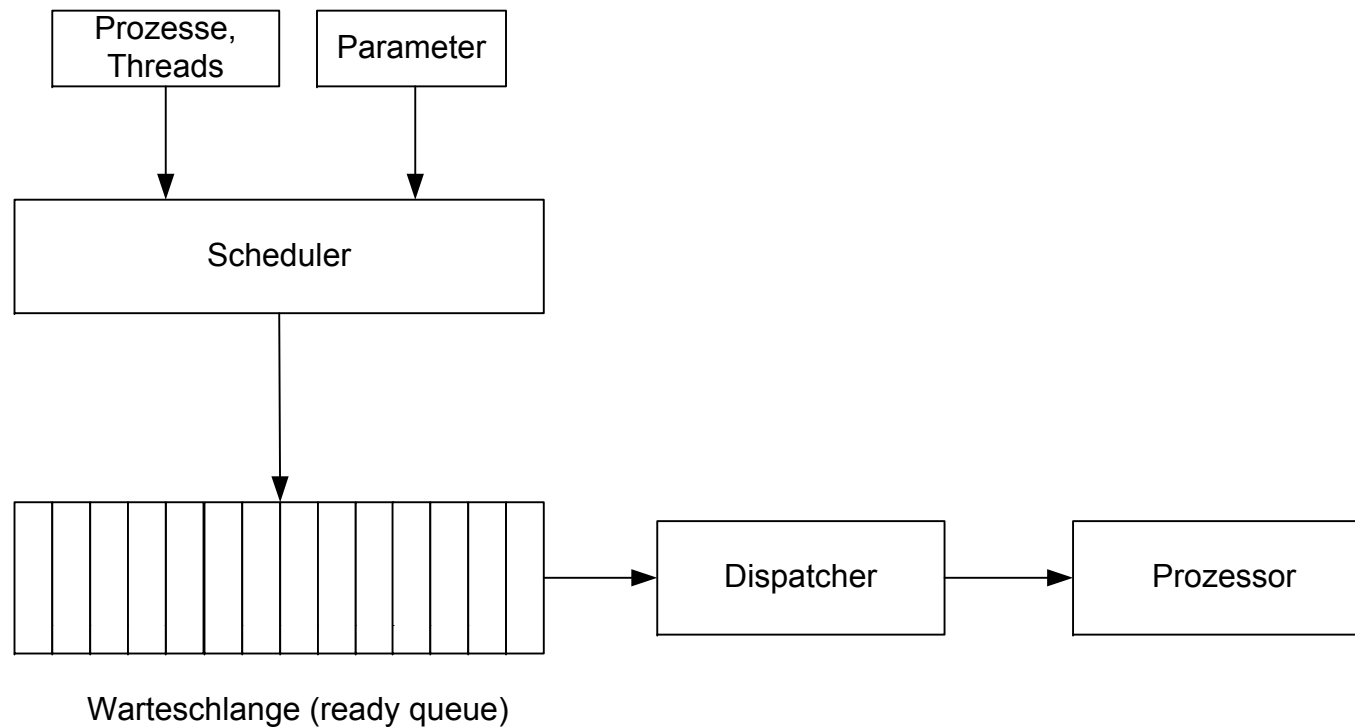
- John A. Stankovic, Marco Spuri, Marco Di Natale, and Giorgio C. Buttazzo: Implications of classical scheduling results for real-time systems. IEEE Computer, Special Issue on Scheduling and Real-Time Systems, 28(6):16–25, June 2005.
- Giorgio C. Buttazzo: Rate Monotonic vs. EDF: Judgement Day (<http://www.cas.mcmaster.ca/~downnd/rtsj05-rmedf.pdf>)
- 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

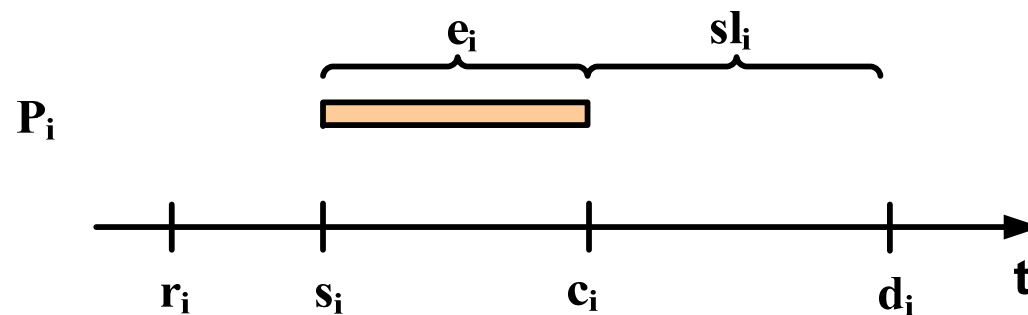


Scheduler und Dispatcher

- **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:
 1. P_i bezeichnet den i. **Prozess** (bzw. Thread)
 2. r_i : **Bereitszeit (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 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 Schedulings 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 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 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.

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 $r_x \geq t_i \wedge d_x \leq t_j$ 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

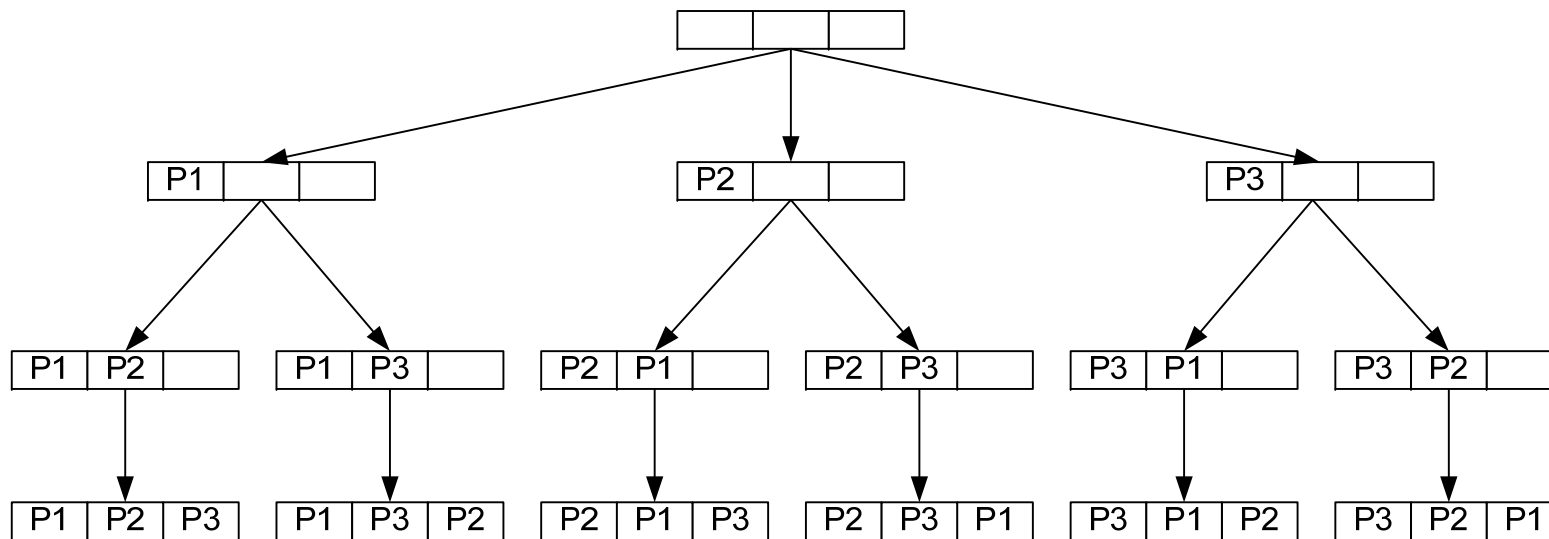


Scheduling

Scheduling-Verfahren für 1-Prozessor-Systeme

Planen durch Suchen

- Betrachtung: ununterbrechbare Aktionen/Prozesse vorausgesetzt
- Lösungsansatz: exakte Planung durch Durchsuchen des Lösungsraums
- Beispiel:
 - $n=3$ Prozesse P_1, P_2, P_3 und 1 Prozessor
 - Suchbaum:



Problem: Komplexität

- $n!$ Permutationen müssen bewertet werden, bei Mehrprozessorsystemen ist das Problem der Planung NP-vollständig
- Durch präemptives Scheduling bzw. durch unterschiedliche Bereitzeiten kann das Problem weiter verkompliziert werden.
- Die Komplexität kann durch verschiedene Maßnahmen leicht verbessert werden:
 - Abbrechen von Pfaden bei Verletzung von Fristen
 - Verwendung von Heuristiken: z.B. Sortierung nach Bereitstellungszeiten r_i
- Prinzipiell gilt jedoch: **Bei komplexen Systemen ist Planen durch Suchen nicht möglich.**

Scheduling-Strategien (online, nicht-präemptiv) für Einprozessorsysteme

1. EDF: Einplanen nach Fristen (Earliest Deadline First): Der Prozess, dessen Frist als nächstes endet, erhält den Prozessor.
2. LST: Planen nach Spielraum (Least Slack Time): Der Prozess mit dem kleinsten Spielraum erhält den Prozessor.
 - Der Spielraum berechnet sich wie folgt:
 $\text{Deadline} - (\text{aktuelle Zeit} + \text{verbleibende Berechnungszeit})$
 - Der Spielraum für den aktuell ausgeführten Prozess ist konstant.
 - Die Spielräume aller anderen Prozesse nehmen ab.
- Vorteil und Nachteile:
 - LST erkennt Fristverletzungen früher als EDF.
 - Für LST müssen die Ausführungszeiten der Prozesse bekannt sein.

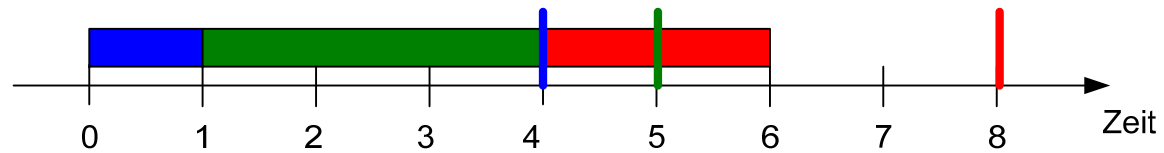
Beispiel

- 3 Prozesse:

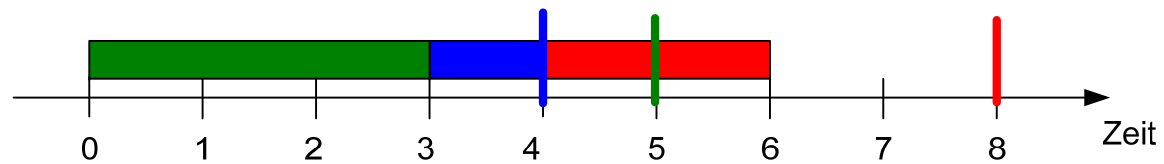
$P_1: r_1=0; e_1=2; d_1=8;$

$P_2: r_2=0; e_2=3; d_2=5;$

$P_3: r_3=0; e_3=1; d_3=4;$



Earliest Deadline First



Least Slack Time

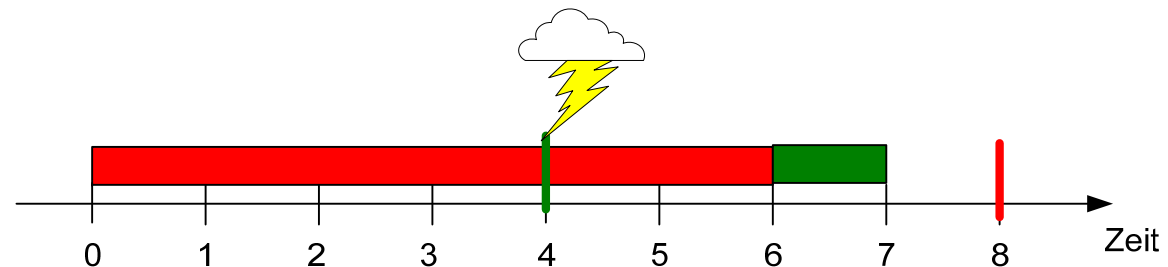
Versagen von LST

- LST kann selbst bei gleichen Bereitzeiten im nicht-präemptiven Fall versagen.

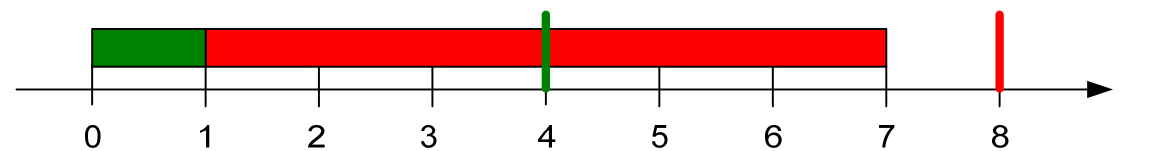
- 2 Prozesse:

$$P_1: r_1=0; e_1=6; d_1=8;$$

$$P_2: r_2=0; e_2=1; d_2=4;$$



LST: P2 verpasst Deadline



EDF liefert optimalen Plan

- Anmerkung: Aus diesem Grund wird LST nur in präemptiven Systemen eingesetzt. Bei Prozessen mit gleichen Spielräumen wird einem Prozess Δ eine Mindestausführungszeit garantiert.

Optimalität von EDF

- Unter der Voraussetzung, dass alle Prozesse P_i eine Bereitzeit $r_i=0$ besitzen und das ausführende System ein Einprozessorsystem ist, ist EDF optimal, d.h. ein zulässiger Plan wird gefunden, falls ein solcher existiert.
- Beweisidee für EDF: Tausch in existierendem Plan
 - Sei Plan_x ein zulässiger Plan.
 - Sei Plan_{EDF} der Plan, der durch die EDF-Strategie erstellt wurde.
 - Ohne Einschränkung der Allgemeinheit: die Prozessmenge sei nach Fristen sortiert, d.h. $d_i \leq d_j$ für $i < j$.
 - Idee: Schrittweise Überführung des Planes Plan_x in Plan_{EDF}
 - $P(\text{Plan}_x, t)$ sei der Prozess, der von Plan_x zum Zeitpunkt t ausgeführt wird.
 - $\text{Plan}_x(t)$ ist der bis zum Zeitpunkt t in Plan_{EDF} überführte Plan ($\Rightarrow \text{Plan}_x(0) = \text{Plan}_x$).

Fortsetzung des Beweises

- Wir betrachten ein Zeitintervall Δ_t .
- Zum Zeitpunkt t gilt:
 $i = P(\text{Plan}_{\text{EDF}}, t)$
 $j = P(\text{Plan}_x, t)$
- Nur der Fall $j > i$ ist interessant. Es gilt:
 - $d_i \leq d_j$
 - $t + \Delta_t \leq d_i$ (ansonsten wäre der Plan_x nicht zulässig)
 - Da die Pläne bis zum Zeitpunkt t identisch sind und P_i im Plan_{EDF} zum Zeitpunkt t ausgeführt sind, kann der Prozess P_i im Plan_x noch nicht beendet sein.
 $\Rightarrow \exists t' > t + \Delta_t: (i = P(\text{Plan}_x, t') = P(\text{Plan}_x, t' + \Delta_t)) \wedge t' + \Delta_t \leq d_i \leq d_j$
 \Rightarrow Die Aktivitätsphase von P_i im Zeitintervall $t' + \Delta_t$ und P_j im Zeitintervall $t + \Delta_t$ können ohne Verletzung der Zeitbedingungen getauscht werden \Rightarrow Übergang von $\text{Plan}_x(t)$ zu $\text{Plan}_x(t + \Delta_t)$