Administratives / Fragen zur letzten Vorlesung

- Administratives:
  - Prüfungsanmeldung ist jetzt freigeschaltet. Die Anmeldung ist bis zum 15.1.2014 möglich.
  - Bitte rechtzeitig anmelden, damit wir entsprechend der Teilnehmerzahlen planen können.
- Fragen zur Vorlesung:
  - Diskussion zum Thema Optimalität:
    - Optimalität im Allgemeinen: wichtig ist vor allem, dass der Entwickler eine Abschätzung machen kann, ob es zu Deadlineverletzungen kommen kann (100% Last ist aus Gründen der Robustheit sowieso zu vermeiden)
    - Optimalität in Multicoresystemen: in Echtzeitsystemen wird im überwiegenden Fall ohnehin eine Zuweisung der Prozesse auch Cores vorgenommen. Insofern reduziert sich das Scheduling-Problem auf die Single-Core Variante (→ ausgenommen der Problematik der Abhängigkeit von Prozessen, sowie der Worst Case Execution Time Analyse, siehe später)

Bewertung

- Die Strategien EDF und LST werden in der Praxis selten angewandt.
- Gründe:
  - Bereitzeiten sind nur bei zyklischen Prozessen oder Terminprozessen bekannt.
  - Häufig liegen nicht einmal konkrete Fristen vor → diese werden erst bei Tests ermittelt
  - Die Abschätzung der für LST benötigten Laufzeit ist sehr schwierig (siehe Exkurs).
  - Die Implementierung erfordert ein Sortieren nach Fristen bzw. Slack Time → zeitaufwändig und generell werden dynamische Datenstrukturen gerne vermieden.
  - Synchronisation, Kommunikation und gemeinsame Betriebsmittel verletzen die Forderung nach Unabhängigkeit der Prozesse.

Ansatz in der Praxis

- Zumeist basiert das Scheduling auf der Zuweisung von (statischen) Prioritäten.
- Prioritäten werden zumeist durch natürliche Zahlen zwischen 0 und 255 ausgedrückt. Die höchste Priorität kann dabei sowohl 0 (z.B. in VxWorks) als auch 255 (z.B. in POSIX) sein.
- Die Priorität ergibt sich aus der Wichtigkeit des technischen Prozesses und der Abschätzung der Laufzeiten und Spielräume. **Die Festlegung erfolgt dabei durch den Entwickler (→ Fehlerquelle).**
- Bei gleicher Priorität wird zumeist eine FIFO-Strategie (d.h. ein Prozess läuft solange, bis er entweder beendet ist oder aber ein Prozess höherer Priorität eintrifft) angewandt.
  Alternative Round Robin: Alle laufbereiten Prozesse mit der höchsten Priorität erhalten jeweils für eine im Voraus festgelegte Zeitdauer die CPU. (Vorteil: robuster gegenüber Laufzeitüberschreitungen, Nachteil: mehr Kontextwechsel)

Klausur SS 07 – Szenario (20 Punkte = 20 min) – leicht modifiziert

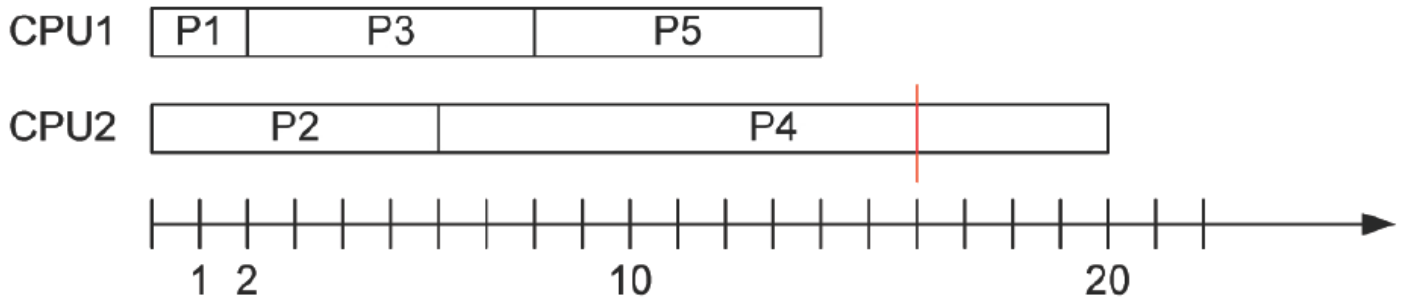
a) Welches Schedulingverfahren wurde verwendet? Welche Änderungen würden sich ergeben, wenn das Verfahren präemptiv wäre?

b) Welche optimalen Schedulingverfahren unter welchen Voraussetzungen existieren für Mehrprozessorsysteme?

c) Zeichnen Sie unter Zuhilfenahme eines optimalen Schedulingverfahrens einen korrekten Ausführungsplan.

d) In der Praxis werden diese Schedulingverfahren nicht angewandt. Was spricht dagegen und welcher Ansatz wird stattdessen gewählt?

Startzeiten s: s(P1)=0; s(P2)=0; s(P3)=0; s(P4)=0; s(P5)=0;
Ausführungszeiten e: e(P1)=2; e(P2)=6; e(P3)=6; e(P4)=14; e(P5)=6;
Deadlines e: e(P1)=4; e(P2)=8; e(P3)=12; e(P4)=16; e(P5)=18;



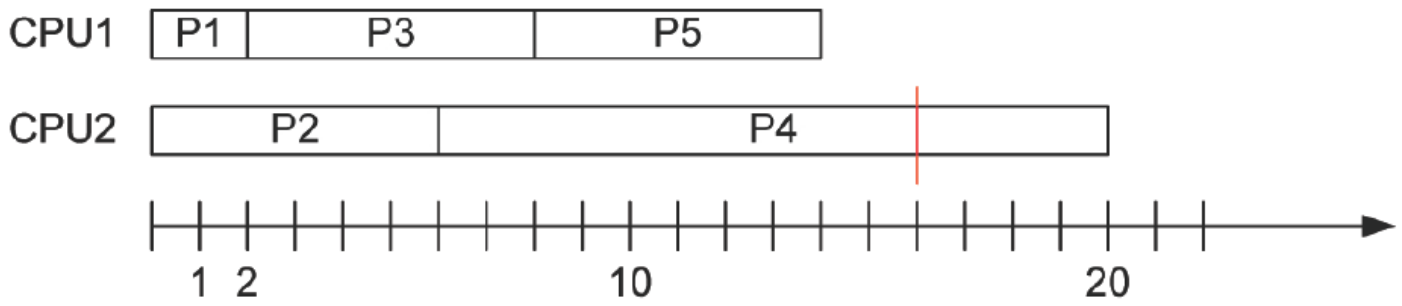
Klausur SS 07 – Antworten

a) EDF, es ergeben sich keine Änderungen, wenn das Verfahren präemptiv ausgeführt wird

b) Präemptives Planen (wenn alle Prozesse inkl. Startzeit, Laufzeit und Deadline vorab bekannt sind), präemptives Least Slack Time (gleiche Voraussetzungen wie planen, aber zusätzlich müssen alle Prozesse die gleiche Startzeit haben)

c) CPU 1: P1-> P4; CPU 2: P2->P3->P5

d) Es werden prioritätenbasierte Verfahren angewandt, da diese sehr leicht umzusetzen sind und zur Laufzeit neben der Priorität keine weiteren Informationen verfügbar sein müssen



Scheduling

Zeitplanen periodischer Prozesse

Zeitplanung periodischer Prozesse

- Annahmen für (präemptives) Scheduling periodischer Prozesse
  - Alle Prozesse treten periodisch mit einer Frequenz f_i auf.
  - Die Frist eines Prozesses entspricht dem nächsten Startpunkt.
  - Sind die maximalen Ausführungszeiten e_i bekannt, so kann leicht errechnet werden, ob ein ausführbarer Plan existiert.
  - Die für einen Prozesswechsel benötigten Zeiten sind vernachlässigbar.
  - Alle Prozesse sind unabhängig.
- Eine sehr gute Zusammenfassung zu dem Thema Zeitplanung periodischer Prozesse liefert Giorgio C. Buttazzo in seinem Paper „Rate Monotonic vs. EDF: Judgement Day“ (http://www.cas.mcmaster.ca/~downd/rtsj05-rmedf.pdf).

Einplanbarkeit

- Eine notwendige Bedingung zur Einplanbarkeit ist die Last:
  - Last eines einzelnen Prozesses: $\rho_i = e_i * f_i$
  - Gesamte Auslastung bei n Prozessen:

$$\rho = \sum_{i=0}^{n} \rho_i$$

  - Bei m Prozessoren ist $\rho < m$ eine notwendige aber nicht ausreichende Bedingung.

Zeitplanen nach Fristen

- **Ausgangspunkt**: Wir betrachten Systeme mit einem Prozessor und Fristen der Prozesse, die relativ zum Bereitzeitpunkt deren Perioden entsprechen, also $d_i=1/f_i$.
- **Aussage**: Die Einplanung nach Fristen ist optimal.
- **Beweisidee**: siehe Folien bei Einführung von EDF
- Leider wird aufgrund von diversen Vorurteilen EDF selten benutzt.
- Betriebssysteme unterstützen selten ein EDF-Scheduling
  → Die Implementierung eines EDF-Scheduler auf der Basis von einem prioritätsbasierten Scheduler ist nicht effizient zu implementieren (Ausnahme: zeitgesteuerte Systeme)

Zeitplanung nach Raten

- Rate Monotonic bezeichnet ein Scheduling-Verfahren mit festen Prioritäten Prio(i), die sich proportional zu den Frequenzen verhalten.

  ⟶ Prozesse mit hohen Raten werden bevorzugt. Das Verfahren ist optimal, falls eine Lösung mit statischen Prioritäten existiert. Verfahren mit dynamischen Prioritäten können allerdings eventuell bessere Ergebnisse liefern.

- Liu und Layland haben 1973 in einer Worst-Case-Analyse gezeigt, dass Ratenplanung sicher erfolgreich ist, falls bei n Prozessen auf einem Prozessor gilt:

$$\rho \leq \rho_{max} = n*(2^{1/n} - 1)$$

$$\lim_{n \to \infty} \rho_{max} = \ln 2 \approx 0{,}69$$

- Bei harmonischen Frequenzen liefert RM bis zu einer Auslastung von 100% gültige Schedules.
- Derzeit zumeist verwendetes Scheduling-Verfahren im Bereich von periodischen Prozessen.

Scheduling

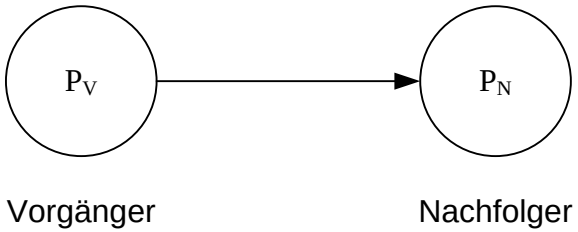
Planen abhängiger Prozesse

Allgemeines zum Scheduling in Echtzeitsystemen

- Grundsätzlich kann der Prozessor neu vergeben werden, falls:
  - ein Prozess endet,
  - ein Prozess in den blockierten Zustand (z.B. wegen Anforderung eines blockierten Betriebsmittels) wechselt,
  - eine neuer Prozess gestartet wird,
  - ein Prozess vom blockierten Zustand in den Wartezustand wechselt (z.B. durch die Freigabe eines angeforderten Betriebmittels durch einen anderen Prozess)
  - oder nach dem Ablauf eines Zeitintervals, siehe z.B. Round Robin.
- Hochpriorisierte Prozesse dürfen in Echtzeitsystemen nicht durch unwichtigere Prozesse behindert werden → Die Prioritätsreihenfolge muss bei allen Betriebsmitteln (CPU, Semaphore, Netzkommunikation, Puffer, Peripherie) eingehalten werden, d.h. Vordrängen in allen Warteschlangen.

Präzedenzsysteme

- Zur Vereinfachung werden zunächst Systeme betrachtet, bei denen die Bereitzeiten der Prozesse auch abhängig von der Beendigung anderer Prozesse sein können.
- Mit Hilfe von Präzedenzsystemen können solche Folgen von voneinander abhängigen Prozessen beschrieben werden.
- Zur Beschreibung werden typischerweise Graphen verwendet:



- Der Nachfolgerprozess kann also frühestens beim Erreichen der eigenen Bereitzeit **und** der Beendigung der Ausführung des Vorgängerprozesses ausgeführt werden.

Probleme bei Präzedenzsystemen

- Bei der Planung mit Präzedenzsystemen muss auch berücksichtigt werden, dass die Folgeprozesse noch rechtzeitig beendet werden können.
- Beispiel:
  P_V: $r_V=0$; $e_V=1$; $d_V=3$;
  P_N: $r_N=0$; $e_N=3$; $d_N=5$;
- Falls die Frist von P_V voll ausgenutzt wird, kann der Prozess P_N nicht mehr rechtzeitig beendet werden.

  $\rightarrow$ Die Fristen müssen entsprechend den Prozessabhängigkeiten neu berechnet werden (Normalisierung von Präzedenzsystemen).

Normalisierung von Präzedenzsystemen

- Anstelle des ursprünglichen Präzedenzsystems PS wird ein normalisiertes Präzedenzsystem PS' mit folgenden Eigenschaften:
  - $\forall i: e'_i = e_i$
  - $\forall i : d'_i = \begin{cases} d_i, falls N_i = \emptyset \\ min(d_i, min(d'_q - e'_q | q \in N_i) \end{cases}$

    wobei N_i die Menge der Nachfolger im Präzedenzgraph bezeichnet und d'_i rekursiv beginnend bei Prozessen ohne Nachfolger berechnet wird.
  - Falls die Bereitzeiten von externen Ereignissen abhängig sind, gilt $r'_i = r_i$. Sind die Bereitzeiten dagegen abhängig von der Beendigung der Prozesse, so ergeben sie sich aus dem konkreten Scheduling.

  eingeführt.

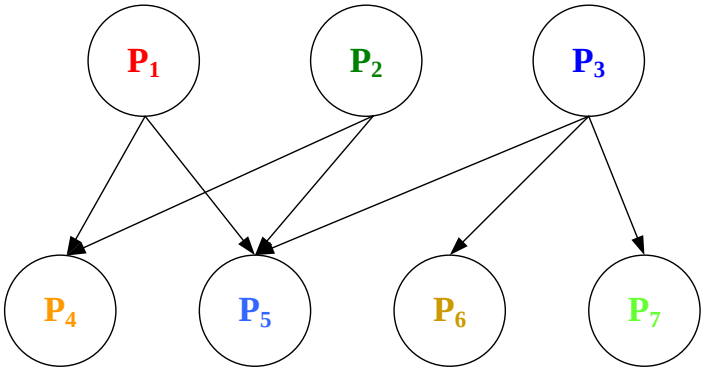
  $\rightarrow$ Ein Präzedenzsystem ist nur dann planbar, falls das zugehörige normalisierte Präzedenzsystem planbar ist.

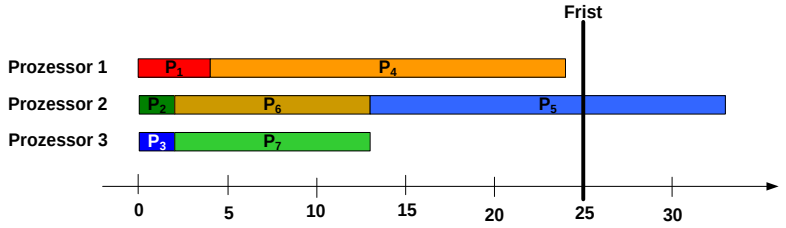
Anomalien bei nicht präemptiven Scheduling

- Wird zum Scheduling von Präzedenzsystemen ein nicht präemptives prioritätenbasiertes Verfahren (z.B. EDF, LST) verwendet, so können Anomalien auftreten:
  - Durch Hinzufügen eines Prozessors kann sich die gesamte Ausführungszeit verlängern.
  - Durch freiwilliges Warten kann die gesamte Ausführungszeit verkürzt werden.

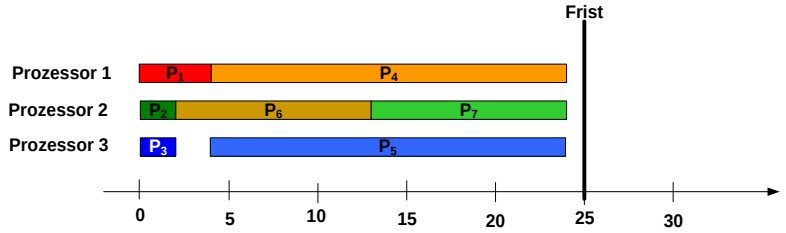
Beispiel: Verkürzung durch freiwilliges Warten

- Beispiel: 3 Prozessoren, 7 Prozesse (r_i=0, e_1=4; e_2=2; e_3=2; e_4=20; e_5=20; e_6=11; e_7=11, d_i=25), Präzedenzgraph:





Prioritätenbasiertes nicht-präemptives Scheduling (z.B. LST)

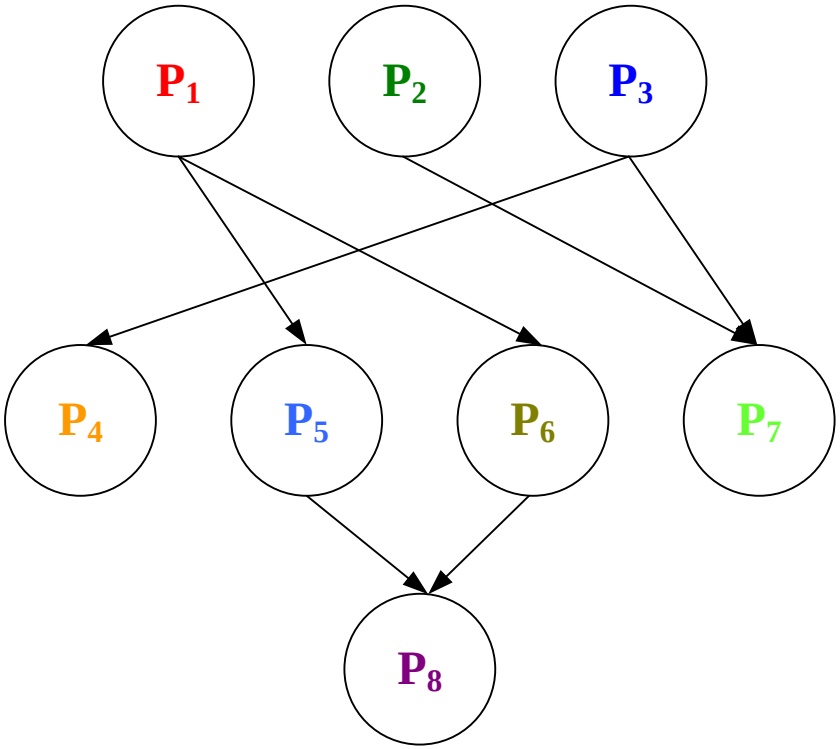


Optimaler nicht-präemptiver Plan

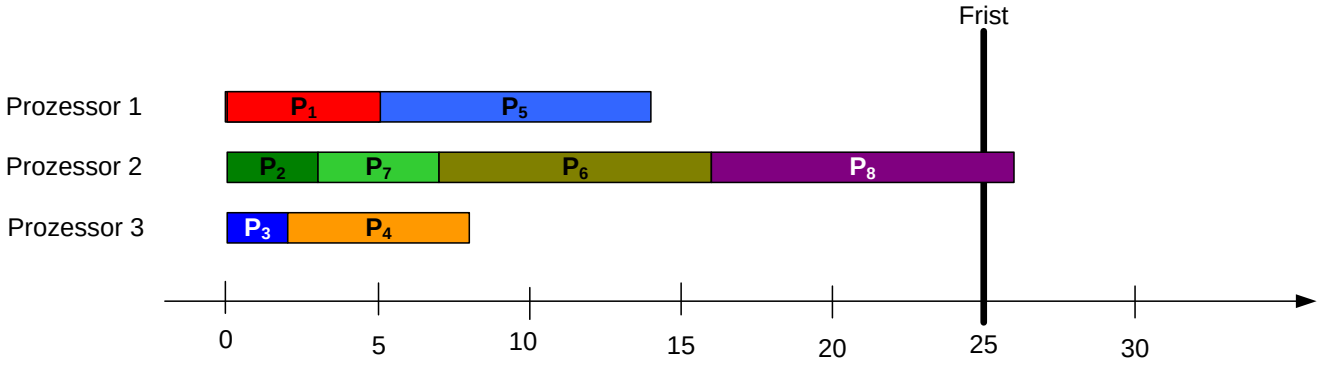
Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II

- Beispiel:
  - 2 bzw. 3 Prozessoren
  - 8 Prozesse:
    - Startzeiten $r_i=0$
    - Ausführungszeiten
      $e_1=5$;
      $e_2=3$;
      $e_3=2$;
      $e_4=6$;
      $e_5=9$;
      $e_6=9$;
      $e_7=4$,
      $e_8=10$
    - Frist: $d_i=25$

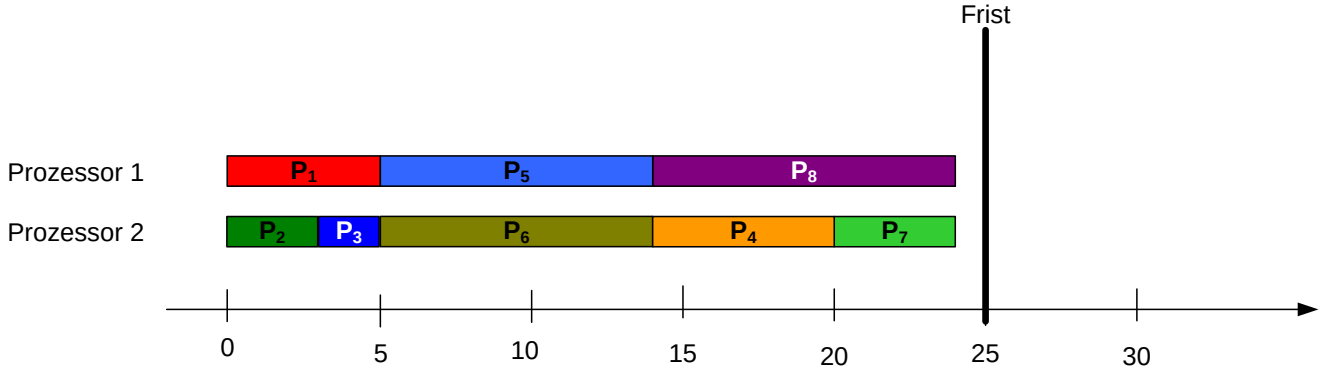
- Präzedenzgraph:



Beispiel: Laufzeitverlängerung durch zusätzlichen Prozessor II



Prioritätenbasiertes nicht-präemptives Scheduling (LST) auf 3 Prozessoren



Prioritätenbasiertes nicht-präemptives Scheduling (LST) auf 2 Prozessoren

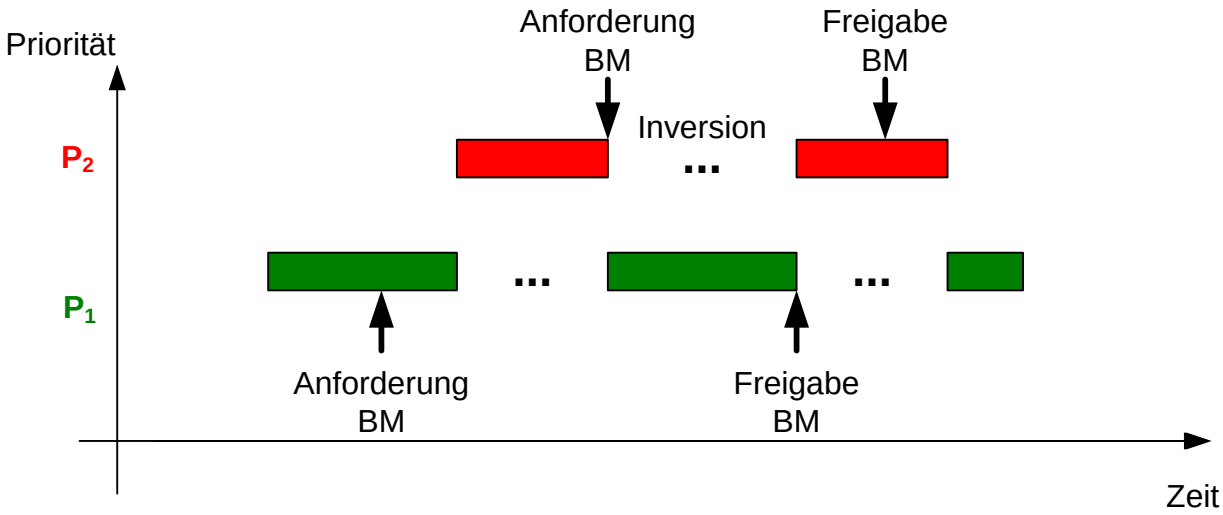
Scheduling

Problem: Prioritätsinversion

Motivation des Problems

- Beim Zugriff auf Betriebsmittel kann es dazu kommen, dass ein niederpriorer Prozess durch den Besitz eines Betriebsmittel einen hochprioren Prozess blockiert (**Begriff: Prioritätsinversion**)
- Beispiel aus dem Alltag:
  - Computer wird vom Bruder zum Spielen blockiert, obwohl man den Computer dringend für Hausaufgaben braucht

Begrenzte Inversion

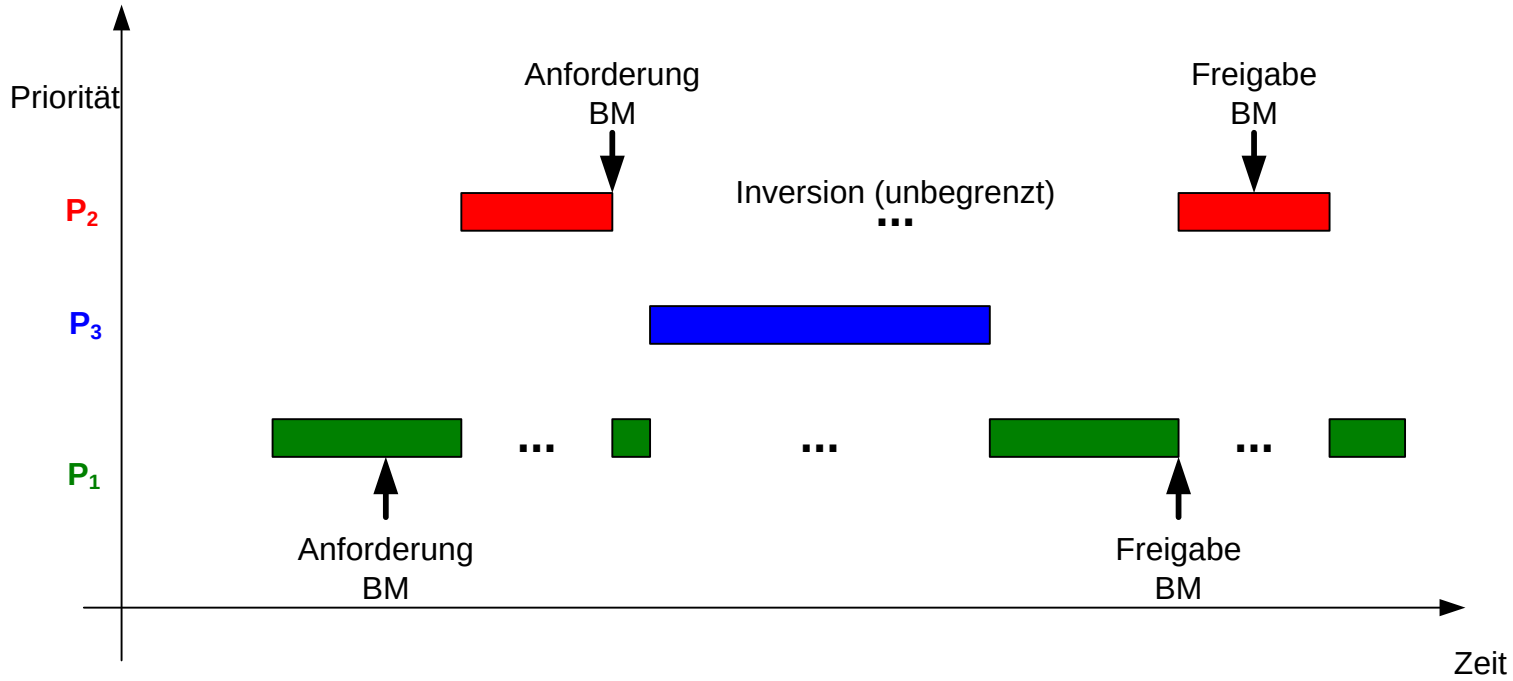


- Dieses Problem ist unvermeidbar, wenn beide Prozesse auf die Ressourcen zugreifen sollen, allerdings können die Auswirkungen durch folgende **Designregel** minimiert werden:
  - Niederpriore Prozesse müssen Ressourcenbelegung so kurz wie möglich halten
  - Die Blockierung durch niederpriore Prozesse muss bei der Zeitabschätzung berücksichtigt werden

Motivation des Problems - Fortsetzung

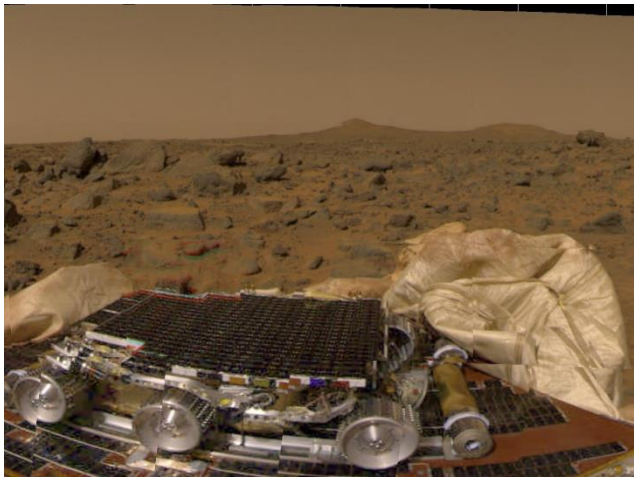
- Trotz Beachtung der vorgenannten Designregel kann es zu einer Verschärfung der Problematik kommen
- Fortsetzung des Beispiels: die Ressourcen können nicht einfach entzogen werden und der niederpriore Prozess wird von einem weiteren, mittelprioren Prozess für eine längere Zeit unterbrochen
  - Beispiel: Computer ist gesperrt, Bruder telefoniert mit seiner Freundin

Unbegrenzte Inversion



Reales Beispiel: Mars Pathfinder (Projektkosten: 175 Mio USD)

- **System**: Der Mars Pathfinder hatte zur Speicherung der Daten einen Informationsbus (vergleichbar mit Shared Memory). Der Informationsbus war durch einen binären Semaphore geschützt. Ein Bus Management Prozess verwaltete den Bus mit hoher Priorität.
  Ein weiterer Prozess war für die Sammlung von geologischen Daten eingeplant. Dieser Task lief mit einer niedrigen Priorität und griff ebenfalls regelmäßig auf den Informationsbus zu. Zusätzlich gab es noch einen Kommunikationsprozess mittlerer Priorität.
- **Symptome**: Das System führte in unregelmäßigen Abständen einen Neustart durch. Daten gingen dadurch verloren.
- **Ursache**: Der binäre Semaphor war nicht mit dem Merkmal zur Unterstützung von Prioritätsvererbung (siehe später) erzeugt worden. Dadurch kam es zur Prioritätsinversion. Ein Watchdog (Timer) erkannte eine unzulässige Verzögerung des Bus Management Prozesses und führte aufgrund eines gravierenden Fehlers einen Neustart durch.



Quelle: Nasa

Ansätze zur Lösung der Prioritätsinversion

- Es existieren verschiedene Ansätze, um das Problem der unbegrenzten Prioritätsinversion zu begrenzen:
  - Nicht-unterbrechbare kritische Bereiche
  - Prioritätsvererbung (priority inheritance)
  - Prioritätsobergrenzen (priority ceiling)
  - Unmittelbare Prioritätsobergrenzen (immediate priority ceiling)
- Anforderungen an Lösungen:
  - Leichte und anwendungsunabhängige Implementierung
  - Minimierung und gute Abschätzung der maximalen Verzögerung (insbesondere für hochpriore Prozesse)
  - Optional: Ausschluss von Verklemmungen

Nicht-unterbrechbare kritische Bereiche (Nonpreemptive critical section)

- Lösung: der Prozess wird für den kritischen Bereich nicht unterbrochen. Die unbegrenzte Prioritätsinversion wird somit verhindert.
- Bewertung der Lösung:
  - \+ einfach und anwendungsunabhängig zu implementieren
  - \+ leichte Abschätzung der maximalen Blockierung
  - \- Verzögert höherpriorisierte Prozesse auch dann, wenn diese das Betriebsmittel nicht benötigen

Prioritätsvererbung (priority inheritance)

- Sobald ein Prozess höherer Priorität ein Betriebsmittel anfordert, das ein Prozess mit niedrigerer Priorität besitzt, erbt der Prozess mit niedrigerer Priorität die höhere Priorität. Nachdem das Betriebsmittel freigegeben wurde, fällt die Priorität wieder auf die ursprüngliche Priorität zurück.
  - **+** Unbegrenzte Prioritätsinversion wird verhindert
  - **+** Einfache, anwendungsunabhängige Implementierung
  - **-** Blockierungen werden hintereinander gereiht (Blockierungsketten)
  - **-** Verklemmungen durch Programmierfehler werden nicht verhindert

Laufendes Beispiel: Ausgangszustand