

# Echtzeitsysteme

## Zugriffskontrolle

**Peter Ulbrich**

Lehrstuhl für Verteilte Systeme und Betriebssysteme  
Friedrich-Alexander-Universität Erlangen-Nürnberg

<https://www4.cs.fau.de>

13. Januar 2016



# Prüfungen



## Terminvereinbarung zur mündlichen Prüfung

- Erfolgt erstmals elektronisch (Poll) → [Windhundverfahren](#)
- Terminbestätigung nach Anmeldeschluss<sup>1</sup>
- ⚠ **Anmeldefrist:** Semesterende (05.02.2016)



## Anmeldevorgang

- Link zur Teilnahme → Mail an alle Angemeldeten<sup>1</sup>
- ⚠ **Teilnehmerformat:** <Nachname, Vorname> EZS<7,5/5>  
(Beispiel: Müller, Klaus EZS7,5 oder Meier, Emelie EZS5)

## ■ Terminliche Probleme, Änderungen und Abmeldung

- Individuelle Terminvereinbarung in Ausnahmefällen möglich
- Rechtzeitige Abmeldung erlaubt uns Reorganisation
- Wir beißen nicht!

<sup>1</sup> Sonst bitte umgehend Mail an uns!



# Organisatorisches



## Evaluation der Veranstaltung

- Eure Meinung ([Lob/Kritik](#)) ist uns wichtig!
- Eure Rückmeldung hat Konsequenzen (z.B. Folien-Redesign)
- Bitte evaluiert [Vorlesung](#) und [Übungen](#)



## ⚠ Typische Rückläuferquote → 2 – 10%

- Zu wenig für eine sinnvolle Einschätzung
- Aber: Typische Rückläuferquote in EZS → 60 – 80%

## Motivationsanreiz zur Evaluation



- **Traditionell:** Kaffee und Kekse in der letzten Vorlesung
- **Harte Bedingung:** ≥ 60% der ausgegebenen TANs werden evaluiert!



# Gliederung

## 1 Überblick

## 2 Konkurrenz und Koordination

- Kausalordnung und Koordinierung
- Konkurrenz und Konflikte

## 3 Synchronisation Considered Harmful

## 4 Echtzeitfähige Synchronisationsprotokolle

- Verdrängungssteuerung
- Prioritätsvererbung
- Prioritätsobergrenzen

## 5 Ablaufplanung

## 6 Zusammenfassung



## Fragestellungen

- **Gegenseitiger Ausschluss** in Echtzeitsystemen?
  - Betriebsmittel und Betriebsmittelarten
  - Konkurrenz, Wettbewerb und Konflikte
  - Einfluss auf das **Laufzeitverhalten**?
- Umgang mit (unvermeidlichen) **Prioritätsumkehr**?
  - Zeitlichen Einfluss kritischer Abschnitte **bestimmen**
  - Ablaufplanung in ereignisgesteuerten Systemen
- **Synchronisationsprotokolle** für Echtzeitsysteme
  - Zeitlichen Einfluss kritischer Abschnitte **begrenzen**
  - Verdrängungssteuerung, Prioritätsvererbung, Prioritätsobergrenzen



**Unkontrollierte Prioritätsumkehr** und **Verklemmungen** vermeiden



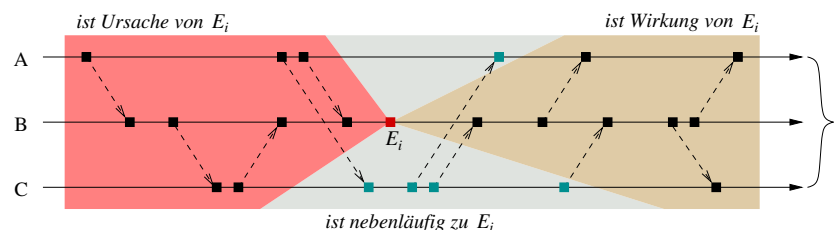
## Gliederung

- 1 Überblick
- 2 Konkurrenz und Koordination
  - Kausalordnung und Koordinierung
  - Konkurrenz und Konflikte
- 3 Synchronisation Considered Harmful
- 4 Echtzeitfähige Synchronisationsprotokolle
  - Verdrängungssteuerung
  - Prioritätsvererbung
  - Prioritätsobergrenzen
- 5 Ablaufplanung
- 6 Zusammenfassung



## Wiederholung: Nebenläufigkeit

Wiederholung von VI/8 ff



- **Relationen:** Ursache  $\leftrightarrow$  Wirkung  $\leftrightarrow$  Nebenläufigkeit



Ein Ereignis  $E_i$  **ist nebenläufig zu** einem anderen:

→ Es liegt im **Anderswo** anderen Ereignisses



**Kausalordnung** von Ereignissen

- **Rangfolge** → zeitlich geordnete Ereignisse (vgl. Kapitel IV)
- **Zugriffskontrolle** → nebenläufige Ereignisse



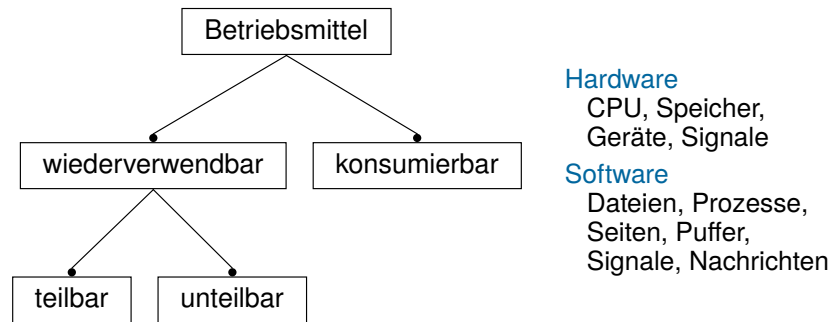
## Koordinierung

Reihenschaltung nebenläufiger Aktivitäten

- **Synchronisation** (gr. *syn* : zusammen, *chrónos* : Zeit)
  - „Herstellen von Gleichzeitigkeit“
  - Koordination der Kooperation und Konkurrenz zwischen Aufgaben
  - Abgleich von Echtzeituhren (oder Daten) in verteilten Systemen
- **Zugriffskontrolle** → Koordinierung nebenläufiger Ereignisse
  - **Synchronisation** gleichzeitiger Zugriffe auf **gemeinsame Betriebsmittel**
  - Herstellen einer Rangfolge für nebenläufige Ereignisse
  - Sequentialisierung von Arbeitsaufträgen entlang einer Kausalordnung
- **Umsetzung der Zugriffskontrolle** (vgl. VI/13 ff)
  - **Implizit** in taktgesteuerten Systemen  $\leadsto$  geeigneter Ablaufplan
  - **Explizit** in ereignisgesteuerten Systemen  $\leadsto$  Synchronisationsprotokolle
  - Analytische versus konstruktive Maßnahmen (vgl. VI/13 ff)



## Betriebsmittel und Betriebsmittelarten



- **Wettbewerb um Betriebsmittel** (engl. *resource contention*) bezieht sich auf Anzahl und Art eines Betriebsmittels

einseitige Synchronisation → konsumierbare Betriebsmittel

mehrseitige Synchronisation → wiederverwendbare Betriebsmittel

- Begrenzung, gegenseitiger Ausschluss



## Konfliktsituation

Blockierung von Arbeitsaufträgen



Arbeitsaufträge befinden sich im **Konflikt**, falls:

- **Begrenzte Anzahl** gemeinsamer Betriebsmitteln verfügbar
- **Gemeinsame Verwendung** derselben, unteilbaren Betriebsmittel

- **Streit** (engl. *contention*) um ein Betriebsmittel

→ Einer will, was der andere hat

- Anfordernder Auftrag **blockiert** und wartet auf die Freigabe des Betriebsmittels durch den belegenden Auftrag
- Belegende Auftrag gibt das Betriebsmittel frei und **deblockiert** den anfordernden Auftrag



Begrenzte/unteilbare Betriebsmittel implizieren **Kooperation**



## Synchronisationsprimitive zur Kooperation

Serialisierung von Arbeitsaufträgen mit begrenzten/unteilbaren Betriebsmitteln



Unteilbare Betriebsmittel können von gleichzeitigen Arbeitsaufträgen nur nacheinander belegt werden

**Vergabe** → Betriebsmittel sperren und dem Auftrag zuteilen ... **P**

- Erneute Belegung eines gesperrtes Betriebsmittel führt zur **Blockierung** des anfordernden Auftrags
- Der blockierende Auftrag erwartet das Ereignis/Signal zur **Freigabe** des gesperrten Betriebsmittels, ihm wird die CPU entzogen

**Freigabe** → Betriebsmittel entziehen/abgeben und freigeben ... **V**

- Nur der das Betriebsmittel **besitzende** Auftrag kann es freigeben
- Wartende Aufträge führen zur sofortigen **Wiedervergabe**:
  - (a) Betriebsmittel entsperren und alle Aufträge deblockieren → Konkurrenz um die Wiedervergabe
  - (b) Betriebsmittel entsperren und einem ausgewählten Auftrag zuteilen



## Gliederung

- 1 Überblick
- 2 Konkurrenz und Koordination
  - Kausalordnung und Koordinierung
  - Konkurrenz und Konflikte
- 3 Synchronisation Considered Harmful
- 4 Echtzeitfähige Synchronisationsprotokolle
  - Verdrängungssteuerung
  - Prioritätsvererbung
  - Prioritätsobergrenzen
- 5 Ablaufplanung
- 6 Zusammenfassung



## Intervalle von Unverdrängbarkeit

Blockierung, Hemmung (engl. *blocking*)

### Kritischer Abschnitt (engl. *critical section*)

- Folge von Anweisungen, deren Ausführung einen gegenseitigen Ausschluss erfordern  $\leadsto$  **mehrseitige Synchronisation**
  - (a) Vor Überlappung schützen  $\rightarrow$  binärer Semaphor
  - (b) Vor Verdrängung schützen  $\rightarrow$  Einlastung abschalten

### ■ Beispiel: Blockierung durch Systemaufrufe

- Unterbindung von Verdrängung bei Ausführung von Systemaufrufen
- Auswirkungen auf das Laufzeitverhalten von Arbeitsaufträgen:
  - Auftrag  $J_l$  läuft und tätigt einen Systemaufruf
    - $\leadsto J_l$  hat eine niedrige Priorität, durchläuft unverdrängbar den Kern
  - Während des Systemaufrufs, wird Job  $J_h$  ereignisbedingt ausgelöst
    - $\leadsto J_h$  hat eine hohe Priorität, wird eingeplant aber nicht eingelastet
  - $J_l$  blockiert bzw. hemmt  $J_h$ , die Priorität von  $J_h$  wird verletzt

⚠ Synchronisation ist **nicht-funktionale Eigenschaft**  
(in diesem Fall der Systemaufrufe)



## Prioritätsumkehr (engl. *priority inversion*)

Auswirkungen auf das Echtzeitverhalten

⚠ **Prioritätsumkehr** [6] ist Folge der Blockierung eines höher durch einen niedriger priorisierten Auftrag

- 1 Der niedrig priorisierte Auftrag durchläuft einen kritischen Abschnitt und wird vom höher priorisierten Auftrag verdrängt
- 2 Der höher priorisierte Auftrag möchte denselben kritischen Abschnitt betreten, wird vom niedrig priorisierten Auftrag jedoch daran gehindert
- 3 Der niedrig priorisierte Job kann weiter ausgeführt werden, obwohl ein höher priorisierter Job wartet

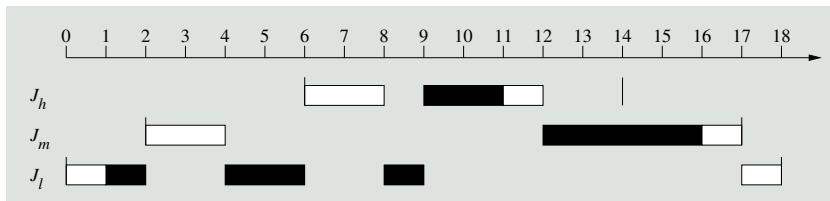
⚠ Diese Form der (**normalen**) **Prioritätsumkehr** ist **nicht vermeidbar**

- Kritischer Abschnitt oder Betriebsmittel: **Unteilbarkeit** ist das Problem



## Beispiel: Wettstreit und Prioritätsumkehr

$J_l \mapsto 6(0, 18]$ ,  $J_m \mapsto 7(2, 17]$ ,  $J_h \mapsto 5(6, 14]$



$J_l$ (niedrige Priorität)	$J_m$ (mittlere Priorität)	$J_h$ (hohe Priorität)
$t_0$ startet	$t_2$ startet, verdrängt $J_l$	$t_6$ startet, verdrängt $J_l$
$t_1$ belegt Betriebsmittel $R$	$t_4$ fordert $R$ an, blockiert	$t_8$ fordert $R$ an, blockiert
$t_4$ setzt Ausführung fort	$t_{12}$ belegt $R$	$t_9$ deblockiert, belegt $R$
$t_8$ setzt Ausführung fort	$t_{16}$ gibt $R$ frei	$t_{11}$ gibt $R$ frei $\leadsto J_m$
$t_9$ gibt $R$ frei $\rightarrow J_h$	$t_{17}$ beendet die Ausführung	$t_{12}$ beendet die Ausführung
$t_{17}$ setzt Ausführung fort		
$t_{18}$ beendet die Ausführung		



## What really happened on Mars?

Prioritätsumkehr beim *Mars Pathfinder* [11, 5]

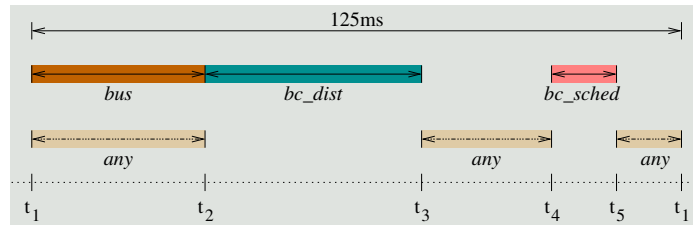
- **bc\_sched**  $\rightarrow$  Task mit höchster Priorität (mit Ausnahme der VxWorks Task „tExec“)
  - Kontrollierte Transaktionen des „1553“-Bus
  - Dieser koppelte Fahr- und Landeeinheit der Raumsonde
- **bc\_dist**  $\rightarrow$  Task mit dritthöchster Priorität
  - Steuerte die Einsammlung der Transaktionsergebnisse
  - Dateneingabe über doppelt gepufferten gemeinsamen Speicher
- **ASI/MET**  $\rightarrow$  Task mit sehr niedriger Priorität
  - Sammelte in seltenen Durchläufen meteorologische Daten ein
  - Interoperierte mit **bc\_dist** (blockierend) auf IPC-Basis

⚠ Hardware gab eine Periodenlänge von 8 Hz (d.h., 125 ms) vor



## What really happened on Mars? (Forts.)

### Aufbau eines Buszyklus



- $t_1$  Transaktion startet hardware-kontrolliert an einer 8 Hz Grenze
- $t_2$  Busverkehr ist zur Ruhe gekommen, bc\_dist wird ausgelöst
- $t_3$  bc\_dist hat die Datenverteilung abgeschlossen
- $t_4$  bc\_sched setzt Transaktion für nächsten Buszyklus auf
- $t_5$  bc\_sched hat seine Aufgabe für diesen Zyklus beendet

Intervalle  $[t_1, t_2]$ ,  $[t_3, t_4]$ ,  $[t_5, t_1]$  standen ASI/MET zur Verfügung



## What really happened on Mars? (Forts.)

### Feste Randbedingung



bc\_dist muss die Datenverteilung vor Auslösung von bc\_sched abgeschlossen haben, um die Transaktion des nächsten Zyklus aufzusetzen:

- Stellt bc\_sched fest, dass bc\_dist noch nicht abgeschlossen ist, wird ein Total-reset durchgeführt
- Der reset hat die Initialisierung der gesamten Hard- und Software zur Folge, insbesondere den **Abbruch aller bodengesteuerten Aktivitäten**
- Bereits aufgezeichnete wiss. Daten sind dann zwar gesichert, aber die noch anstehende Tagesarbeit kann nicht mehr vollbracht werden

- Kategorie **feste Echtzeit** (engl. *firm real-time*) zur Erinnerung:

- **Ergebnis** einer zu einem vorgegebenen Termin nicht geleisteten Arbeit **ist wertlos und wird verworfen**
- Terminverletzung ist tolerierbar, führt zum Arbeitsabbruch



## What really happened on Mars? (Forts.)

### Eine Fehleranalyse



ASI/MET (niedrige Priorität) hat bc\_dist (hohe Priorität) blockiert:

- ASI/MET belegt wiederverwendbares, unteilbares Betriebsmittel
  - Wurde von bc\_dist angefordert, bevor ASI/MET es wieder frei gab
- Im weiteren Verlauf verdrängten Tasks mittlerer Priorität ASI/MET
  - Verlängerung der **Blockierungszeit** für bc\_dist
  - bc\_dist war noch nicht abgeschlossen als bc\_sched startete
- bc\_sched stellte die Zeitverletzung fest und löste einen reset aus

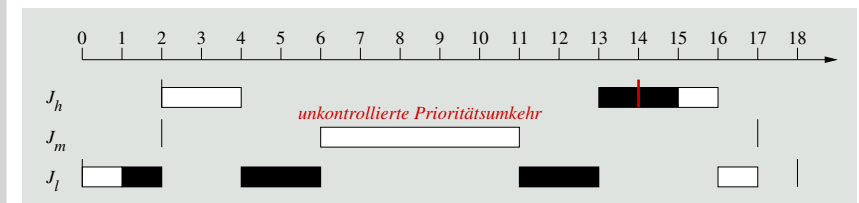
- Fehlererkennung und -beseitigung:

- Die Semaphorinitialisierung war in VxWorks falsch eingestellt
- Sie wurde bodengesteuert (durch ein Skriptprogramm) korrigiert
  - Der Semaphor wurde auf **Prioritätsvererbung** umgestellt



## Anomalie im Laufzeitverhalten

Beispiel:  $J_l \mapsto 7(0, 18]$ ,  $J_m \mapsto 5(2, 17]$ ,  $J_h \mapsto 5(2, 14]$ ;  $J_m$  fordert  $R$  nicht an



- | $J_l$ (niedrige Priorität)           | $J_m$ (mittlere Priorität)      | $J_h$ (hohe Priorität)                 |
|--------------------------------------|---------------------------------|--|
| $t_0$ startet                        | $t_6$ startet, verdrängt $J_l$  | $t_2$ startet, verdrängt $J_l$         |
| $t_1$ belegt Betriebsmittel $R$      | $t_{11}$ beendet die Ausführung | $t_4$ fordert $R$ an, blockiert        |
| $t_4$ setzt Ausführung fort          |                                 | $t_{13}$ belegt $R$                    |
| $t_{11}$ setzt Ausführung fort       |                                 | $t_{14}$ <b>verletzt seinen Termin</b> |
| $t_{13}$ gibt $R$ frei $\mapsto J_h$ |                                 | $t_{15}$ gibt $R$ frei                 |
| $t_{16}$ setzt Ausführung fort       |                                 | $t_{16}$ beendet die Ausführung        |
| $t_{17}$ beendet die Ausführung      |                                 |  |



## Synchronisation *Considered Harmful*

Prioritätsumkehr bei prioritätsorientierter Einplanung

- **Prioritätsumkehr** (siehe Folie 14) ist ein mögliches Phänomen abhängiger Aufträge, welches in zwei Ausprägungen auftreten kann:

### 1 (Normale) Prioritätsumkehr

- Gegenseitiger Ausschluss  $\rightarrow$  Ein Auftrag hoher Priorität wartet auf einen niedrigeren Priorität (ggf. unvermeidbar)

### 2 Unkontrollierte Prioritätsumkehr

- Der Auftrag niedriger Priorität wird von unbeteiligten Aufträgen mittlerer Priorität verdrängt



Lösungsansätze für die blockierende Synchronisation sind:

- Verdrängungssteuerung 👉 Folie 23 ff.
- Prioritätsvererbung 👉 Folie 28 ff.
- Prioritätsobergrenzen 👉 Folie 32 ff.



## Gliederung

- 1 Überblick
- 2 Konkurrenz und Koordination
  - Kausalordnung und Koordinierung
  - Konkurrenz und Konflikte
- 3 Synchronisation Considered Harmful
- 4 Echtzeitfähige Synchronisationsprotokolle
  - Verdrängungssteuerung
  - Prioritätsvererbung
  - Prioritätsobergrenzen
- 5 Ablaufplanung
- 6 Zusammenfassung



## Verdrängungssteuerung – NPCS

Verdrängungsfreie kritische Abschnitte (engl. *non-preemptive critical sections*)



Verdrängung wird für die **Gesamtzeit der Belegung** von (unteilbaren) Betriebsmitteln unterbunden

- Die Benutzung der Betriebsmittel kontrolliert ein **Monitor** [3, 4]

- Kernelized monitor [8], RES\_SCHEDULER in OSEK, ...

**Eintrittsprotokoll**  $\rightarrow$  Verdrängung abwehren

- Ausgelöste Aufträge einplanen, aber nicht einlasten

**Austrittsprotokoll**  $\rightarrow$  Verdrängung wieder zulassen

- Höher priorisierte Aufträge (nachträglich) einlasten



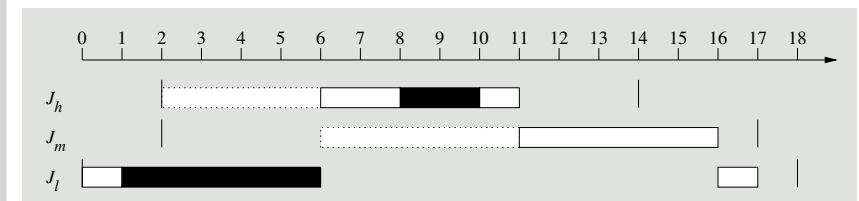
Aufträge **unverdrängbar** bis zur Freigabe des Betriebsmittels

- **Verklemmungsfreies Verfahren** durch **Verklemmungsvorbeugung** (engl. *deadlock prevention*)



## Beispiel: Verdrängungssteuerung

Beispiel (vgl. Folie 20):  $J_l \mapsto 7(0, 18]$ ,  $J_m \mapsto 5(6, 17]$ ,  $J_h \mapsto 5(2, 14]$



**$J_l$  (niedrige Priorität)**

- $t_0$  startet
- $t_1$  belegt  $R$  unverdrängbar
- $t_6$  gibt  $R$  frei  $\rightarrow J_h$
- $t_{16}$  setzt Ausführung fort
- $t_{17}$  beendet die Ausführung

**$J_m$  (mittlere Priorität)**

- $t_6$  wird ausgelöst
- $t_{11}$  startet
- $t_{16}$  beendet die Ausführung

**$J_h$  (hohe Priorität)**

- $t_2$  wird ausgelöst
- $t_6$  startet
- $t_8$  belegt  $R$  unverdrängbar
- $t_{10}$  gibt  $R$  frei
- $t_{11}$  beendet die Ausführung





## Verdrängungssteuerung: Blockierungszeit

Feste obere Schranke

- ☞ **Verzögerungen** nebenläufiger Arbeitsaufträge durch die Zugriffskontrolle **sind nach oben begrenzt**
  - Höher priorisierte Aufträge werden schlimmstenfalls **einmal** durch einen niedriger priorisierten Auftrag blockiert
  - Feste obere Schranke  $b^{rc}$  (*resource contention*) bestimmt sich aus der größten WCET aller kritischen Abschnitte aller niedriger priorisierten Aufträge:  $\max(cs)$
- NPCS verzögert eine periodische Aufgabe  $T_i$  von  $n$  periodischen Aufgaben um  $b_i^{rc} = \max_k(cs_k)$ , für  $i + 1 \leq k \leq n$ :
  - fixed-priority* bei Abarbeitung nach absteigender Priorität
  - dynamic-priority* EDF: Jobs in  $T_i$  mit relativem Termin  $D_i$  können nur durch Jobs mit längeren relativen Terminen als  $D_i$  blockiert werden ☞  $i < j$  wenn  $D_i < D_j$



## NPCS: Ein pragmatischer Ansatz

Effektiv, bei vergleichsweise geringem Aufwand

- ☞ **Kein à priori Wissen** über Betriebsmittelanforderungen **notwendig**
  - Beugt **unkontrollierter** Prioritätsumkehr vor
    - Hält  $J_i$  Betriebsmittel, wird  $J_h$  lediglich eingeplant und blockiert direkt
    - Beendet  $J_i$  seinen kritischen Abschnitt, sind alle Betriebsmittel frei
    - Aufträge geringerer Priorität als  $J_h$  können ihm diese nicht streitig machen
  - Beugt **Verklemmung** (engl. *deadlock*) vor, da Nachforderungen von Betriebsmitteln implizit unteilbar
    - Entkräftet **notwendige** Verklemmungsbedingung [9, VIII-60]
    - „Hold and wait“ Fall kann nicht eintreten
- Einfaches und gutes Verfahren
  - Falls alle **Belegungszeiten** aller Betriebsmittel **kurz** sind
  - Wenn die **meisten** Aufträge **im Konflikt zueinander** stehen
  - Für Systeme mit fester und dynamischer Priorität geeignet



## Pragmatischer Ansatz mit Schönheitsfehlern

Alternativen, sofern bestimmte Voraussetzungen gegeben sind

- ⚠ **Benachteiligt** an Betriebsmitteln **unbeteiligte Arbeitsaufträge**
  - Blockierung hochpriorer Aufträge ohne Konfliktsituation
  - Im Beispiel (vgl. Folie 24) wird  $J_m$  durch  $J_i$  blockiert, obwohl beide Aufträge nicht im gegenseitigen Ausschluss zueinander stehen
- ☞ **Verbesserungsmöglichkeiten**
  - 1 So anderweitige Vermeidung oder Vorbeugung von Verklemmungen:
    - Hochsetzen der Priorität des ein Betriebsmittel haltenden Auftrags für die restliche Belegungszeit auf die Priorität des anfordernden Auftrags
    - Beschleunigung kritischer Abschnitte durch **Prioritätsvererbung** (siehe Folie 32)
  - 2 So Betriebsmittelanforderungen **à priori** bekannt sind:
    - Der ein Betriebsmittel haltende Auftrag läuft mit der höchsten Priorität aller Aufträge, die das Betriebsmittel beanspruchen
    - Das Betriebsmittel besitzt eine **Prioritätsobergrenze** (siehe Folie 32)



## Prioritätsvererbung

Wechsel zwischen zugewiesene und aktuelle (geerbte) Priorität

- ☞ **Prioritätsvererbung** (engl. *priority inheritance*)
  - Priorisierung für die **Restzeit der Belegung** im Konfliktfall
- Vererbung der Priorität durch anfordernde Aufträge
  - Bei Anforderung eines gesperrtes Betriebsmittels, Vererbung der Priorität an den das Betriebsmittel haltenden Auftrag
    - Anfordernde Auftrag hat zu dem Zeitpunkt die **höchste Priorität**
    - Blockierung an dem gesperrten kritischen Abschnitt
    - Priorität des das Betriebsmittel haltenden Auftrags wird erhöht
  - Bei Freigabe des Betriebsmittels, nimmt der niederpriorer Auftrag die ihm ursprünglich zugewiesene Priorität wieder an
    - Der freigebende Auftrag wird ggf. sofort verdrängt
    - Der auf die Freigabe wartende Auftrag wird ggf. sofort eingelastet
- ⚠ **Prioritätsumkehr** wird nicht vermieden, jedoch entschärft:
  - Verdrängbarkeit durch unbeteiligte Arbeitsaufträge höherer Priorität







## Regelwerk – Prioritätsobergrenzen

Betriebsmittelvergabe und Prioritätsvererbung

➡ **Betriebsmittelvergabe** von  $R$  zum Zeitpunkt  $t$  an  $J$  hängt vom Zustand von  $R$  und der aktuellen Priorität  $P(t)$  von  $J$  ab:

**belegt**  $\rightarrow R$  ist gesperrt,  $J$  blockiert

**frei**  $\rightarrow R$  wird  $J$  zugeteilt und gesperrt, falls...

- 1  $P(t) > \hat{\Pi}(t)$ :  $J$ 's Priorität ist größer als die Systemobergrenze
- 2  $P(t) \leq \hat{\Pi}(t)$ :  $J$  ist ein Job, der zum Zeitpunkt  $t$  mindestens ein Betriebsmittel mit Prioritätsobergrenze  $\hat{\Pi}(t)$  hält  
 $\leadsto$  Anderenfalls bleibt  $R$  frei und  $J$  blockiert (siehe Folie 35)

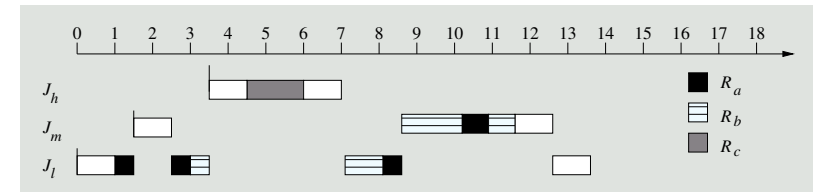
➡ **Prioritätsvererbung** findet auch hier bei Suspendierung statt

- $J_i$  erbt die aktuelle Priorität  $P_h(t)$  von  $J_h$
- $J_i$  behält diese Priorität, bis er alle Betriebsmittel freigibt, deren Prioritätsobergrenze größer oder gleich  $P_h(t)$  ist  
 – Er nimmt dann wieder die Priorität bei der Betriebsmittelzuteilung an



## Beispiel: Prioritätsobergrenzen

Beispiel:  $J_i \mapsto 3.5(0, 18]$ ,  $J_m \mapsto 5(1.4, 17]$ ,  $J_h \mapsto 5(3.5, 14]$



- $J_i$  startet bei  $t_0$  und belegt  $R_a$  an  $t_1$
- $J_m$  verdrängt  $J_i$  an  $t_{1.5}$
- An  $t_{2.5}$  will  $J_m$   $R_b$  belegen  $\leadsto$  Verweigerung, obwohl  $R_b$  frei:  $\hat{\Pi}(t) = P_m$ , aber  $J_i$  belegt bereits  $R_a$  und hat  $\hat{\Pi}(t) = P_m$  verursacht (siehe Folie 33, Regel 2)
- $J_i$  belegt  $R_b$  an  $t_3$
- ...



## Verklemmungsvorbeugung

Entkräftung der hinreichenden Bedingung [9, VIII-60]: zirkulares Warten

➡ Betriebsmittelvergabe durch Prioritätsobergrenzen ist weniger **gefräßig** (engl. *greedy*) als Prioritätsvererbung<sup>2</sup>

- Anforderung von  $J$  kann zurückgewiesen werden, obwohl das angeforderte Betriebsmittel  $R$  frei ist
- Falls die durch die Menge von Prioritätsobergrenzen definierte (ansteigende) **lineare Ordnung** verletzt werden sollte  
 –  $P(t) \leq \hat{\Pi}(t)$  trifft zu und  $J$  hält kein Betriebsmittel mit  $\hat{\Pi}(t)$   
 $\rightarrow$  Die direkte/indirekte Priorität von  $J$  durchbricht die Ordnung
- Alle Betriebsmittel des Systems sind linear geordnet aufgestellt
- Blockierung durch Prioritätsobergrenzen wird auch als **Aufhebungssperre** (engl. *avoidance blocking*) bezeichnet
  - Durch die Verklemmungsvorbeugung implizit anfallende Kosten
  - Blockierung von hochprioren Aufgaben  $J_h$  durch Regel 2, Folie 33



<sup>2</sup>Erinnerung: Verklemmungen werden durch Prioritätsvererbung nicht verhindert.

## Vereinfachung durch Stapelorientierung

Stapelbezogene Einplanung (engl. *stack-based scheduling*)

- Ausgangspunkt ist die **stapelorientierte Einplanung von Prozessen** [1, 2]
  - Nicht immer ist es möglich, jeden Auftrag durch einen eigenen Faden zu repräsentieren bzw. mit einem eigenen Stapel zu versehen  
 – Zu hohe Anzahl an Aufträgen und/oder zu wenig Speicherplatz
  - Gemeinsame Nutzung desselben Stapels setzt voraus, dass kein Auftrag bei Anforderung eines gemeinsamen Betriebsmittels blockiert  
 – Ansonsten droht die Überschreibung der Stapelbereiche anderer Aufträge
  - **Arbeitsaufträge dürfen ihre Ausführung niemals selbst aussetzen**, sie dürfen jedoch von höher priorisierten Aufträgen verdrängt werden  
 – Oben auf dem Stapel läuft immer der Auftrag mit der höchsten Priorität  
 – Logische Konsequenz bei ausgeschlossener Selbstaussetzung



Vereinfachte **stapelbezogene Prioritätsobergrenzen** (engl. *stack-based priority ceiling*)



## Stapelbezogene Prioritätsobergrenzen

Vereinfachtes Reglwerk

- Aktualisierung der **Priorität  $P(t)$**  (siehe Folie 32)
  - Erfolgt mit **jeder Vergabe/Freigabe** von Betriebsmitteln
  - Entspricht einer vorauseilenden Vererbung
- **Einplanung und Einlastung** von Arbeitsaufträgen
  - Nach Auslösung muss ein Auftrag ggf. solange warten, bis die ihm zugewiesene Priorität die Grenzpriorität übersteigt
  - Aufträge werden jeweils entsprechend ihrer zugewiesenen Priorität und verdrängend ausgeführt
- **Zuteilung** eines Betriebsmittels
  - **Erfolgt sofort** mit der Anforderung des Betriebsmittels
- ⚠ **Aufträge blockieren niemals nach Ausführungsbeginn**
  - Ohne Betriebsmittelzuteilung werden Aufträge nicht eingelastet
  - Im Gegensatz zu „normalen“ Prioritätsobergrenzen (siehe Folie 33)



pu Echtzeitsysteme (WS 15/16) – Kapitel VII Zugriffskontrolle  
5 Echtzeitfähige Synchronisationsprotokolle – 5.3 Prioritätsobergrenzen

37/51

## Stapelbezogene Prioritätsobergrenzen (Forts.)

Implikationen

- ☞ Verklemmungen sind durch eine **indirekte Methode zur Verklemmungsvorbeugung<sup>3</sup>** ausgeschlossen
- (a) Mit Ausführungsbeginn einer Aufgabe sind alle im weiteren Verlauf benötigten Betriebsmittel frei
  - Sonst wäre die Grenzpriorität größer oder gleich ihrer Priorität
  - In diesem Fall wäre jedoch die Einlastung verzögert worden
- (b) Bei Verdrängung eines Auftrags sind alle von ihm benötigten Betriebsmittel (noch oder bereits wieder) frei
  - Sonst hätte die Grenzpriorität eine Verdrängung unterbunden
  - Der verdrängende Job wird also immer komplett durchlaufen können
- (c) Auf ein benötigtes Betriebsmittel kann direkt zugegriffen werden



<sup>3</sup>zirkulares Warten wird vorgebeugt (siehe auch Folie 35).

pu Echtzeitsysteme (WS 15/16) – Kapitel VII Zugriffskontrolle  
5 Echtzeitfähige Synchronisationsprotokolle – 5.3 Prioritätsobergrenzen

38/51

## Prioritätsobergrenzen mit dynamischer Priorität

Vergleiche Folie IV-1/27

- ☞ PCP mit festen Prioritäten vergleichsweise einfach
- ⚠ **Dynamische Priorität** → Prioritäten der Aufgaben ändern sich
  - **Grenzprioritäten der Betriebsmittel ändern sich mit der Zeit**
- Aktualisierung der Obergrenzen bei **jeder** Auslösung
  - 1 Dem ausgelösten Auftrag eine Priorität zuweisen
    - Relativ zu den anderen bereits eingeplanten/laufbereiten Jobs
    - Prioritätsorientierte Einplanung je nach Verfahren
  - 2 Grenzprioritäten aller Betriebsmittel aktualisieren
    - Auf Basis des neuen Prioritätsgefüges
  - 3 Grenzpriorität des Systems aktualisieren
    - Auf Basis der neuen Grenzprioritäten der Betriebsmittel
- Für, auf Jobebene, statische oder dynamische Prioritäten



pu Echtzeitsysteme (WS 15/16) – Kapitel VII Zugriffskontrolle  
5 Echtzeitfähige Synchronisationsprotokolle – 5.3 Prioritätsobergrenzen

39/51

## Blockierungszeit

☞ Zugriffskontrolle durch Prioritätsobergrenzen impliziert nunmehr **drei Arten der Blockierung**:

- 1 Direkte Blockierung,
  - 2 Blockierung durch Vererbung,
  - 3 Blockierung durch Aufhebungssperre
- } **Prioritätsvererbung**

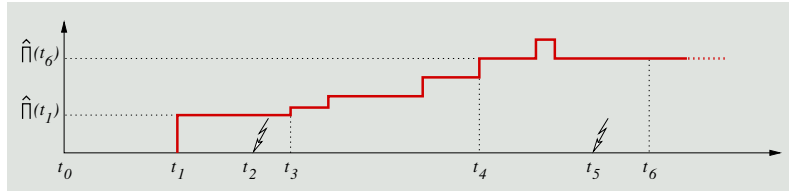
- Effekt von 3. ist, dass jeder Arbeitsauftrag höchstens einmal blockiert und dass eine Blockierung nicht transitiv ist [10]
  - Die Blockierungszeit ist begrenzt durch die größte WCET aller kritischen Abschnitte aller niedriger priorisierten Aufträge
  - Unabhängig von der Anzahl der im Konflikt stehenden Aufträge
    - (a) Wenn ein Auftrag blockiert, dann nur durch höchstens einen Auftrag
    - (b) Ein Auftrag, der einen anderen blockiert, wird selbst nicht blockiert
  - Blockierungszeit  $b_i^{rc} = \max_k (cs_k)$  analog zu NPCS (s. Folie 25)
    - Nur niederpriorige Aufträge  $J_{i+1}, \dots, J_n$  blockieren  $J_i$ :  $i + 1 \leq k \leq n$
    - $J_i$  blockiert aber keine unbeteiligten Aufträge höherer Priorität



pu Echtzeitsysteme (WS 15/16) – Kapitel VII Zugriffskontrolle  
5 Echtzeitfähige Synchronisationsprotokolle – 5.3 Prioritätsobergrenzen

40/51

## Blockierungszeit (Forts.)



$J_l$ (niedrige Priorität)	$J_m$ (mittlere Priorität)	$J_h$ (hohe Priorität)
$t_0$ startet	$t_2$ verdrängt $J_l$	$t_5$ verdrängt $J_m$
$t_1$ belegt $R_x \leadsto \hat{\pi}(t_1)$	$t_3$ belegt $R_y, \pi_m > \hat{\pi}(t_1)$	$t_6$ blockiert

- Direkte Blockierung von  $J_h$  durch  $J_l$  ist nicht möglich: (a) Sonst wäre  $\hat{\pi}(t_3)$  wenigstens  $\pi_h$  und (b)  $J_m$  könnte  $R_y$  überhaupt nicht belegen
- Werden alle ab  $t_2$  angeforderten Betriebsmittel zum Zeitpunkt  $t_6$  nur von  $J_m$  belegt, kann  $J_h$  nur durch  $J_m$  blockiert werden
- Würde  $J_k$  bei  $t_4$  Betriebsmittel  $R_z$  belegen, wäre  $J_h$  aus demselben Grund nicht mehr durch  $J_m$  blockierbar, wie  $J_h$  nicht durch  $J_l$  blockierbar ist

## Gliederung

- 1 Überblick
- 2 Konkurrenz und Koordination
  - Kausalordnung und Koordinierung
  - Konkurrenz und Konflikte
- 3 Synchronisation Considered Harmful
- 4 Echtzeitfähige Synchronisationsprotokolle
  - Verdrängungssteuerung
  - Prioritätsvererbung
  - Prioritätsobergrenzen
- 5 Ablaufplanung
- 6 Zusammenfassung

## Restriktionen des periodischen Modells

Weitere Lockerung von A7 zugunsten mehrseitiger Synchronisation

⚠ Mathematische Ansätze zur zeitlichen Analyse periodischer Echtzeitsysteme bedingen häufig **starke Einschränkungen**:

- A1 ~~Alle Aufgaben sind periodisch~~
- A2 ~~Alle Arbeitsaufträge können an ihren Auslösezeitpunkten eingeplant und ausgeführt werden~~
- A3 Termine und Perioden sind identisch
- A4 Kein Arbeitsauftrag gibt die Kontrolle über den Prozessor ab
- A5 ~~Alle Aufgaben sind unabhängig<sup>4</sup>~~
- A6 Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar
- A7 **Alle Aufgaben verhalten sich voll-präemptiv**

<sup>4</sup>D.h. die einzige gemeinsame Ressource ist die CPU und es existieren keine Einschränkungen hinsichtlich der Auslösezeiten der Arbeitsaufträge voneinander.

## Fadensynchronisation $\leadsto$ Blockierungszeit

Die Blockierungszeit verzögert die Fertigstellung von Arbeitsaufträgen

- Blockierungszeit  $b_i^{rc}$  durch Zugriffskontrolle hängt vom Synchronisationsprotokoll ab:

$$\begin{aligned} \text{NPCS (s. Folie 25)} \quad b_i^{rc} &= \max_{i+1 \leq k \leq n} (cs_k) \\ \text{Priority Inheritance (s. Folie 31)} \quad b_i^{rc} &= \min(n, k) \max_{i+1 \leq l \leq n} (cs_l) \\ \text{Priority Ceiling (s. Folie 40)} \quad b_i^{rc} &= \max_{i+1 \leq k \leq n} (cs_k) \end{aligned}$$

☞ Die tatsächliche Blockierungszeit  $b_i$  schließt Blockierung  $b_i^{np}$  durch nicht-präemptive Bereiche ein:

- Bedingt durch die technische Umsetzung der Zugriffskontrolle
- CPU ist (verdeckte) Ressource

$$\begin{aligned} \text{NPCS} \quad b_i &= \max(b_i^{rc}, b_i^{np}) \\ \text{Priority Inheritance} \quad b_i &= \min(n, k)(b_i^{rc} + b_i^{np}) \\ \text{Priority Ceiling} \quad b_i &= b_i^{rc} + b_i^{np} \end{aligned}$$

## Restriktionen des periodischen Modells

Aufhebung von A4 zugunsten blockierender Synchronisation

⚠ Mathematische Ansätze zur **zeitlichen Analyse** periodischer Echtzeitsysteme bedingen häufig **starke Einschränkungen**:

~~A1 Alle Aufgaben sind periodisch~~

~~A2 Alle Arbeitsaufträge können an ihren Auslösezeitpunkten eingeplant und ausgeführt werden~~

**A3** Termine und Perioden sind identisch

**A4** ~~Kein Arbeitsauftrag gibt die Kontrolle über den Prozessor ab~~

~~A5 Alle Aufgaben sind unabhängig<sup>5</sup>~~

**A6** Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar

~~A7 Alle Aufgaben verhalten sich voll-präemptiv~~

<sup>5</sup>D.h. die einzige gemeinsame Ressource ist die CPU und es existieren keine Einschränkungen hinsichtlich der Auslösezeiten der Arbeitsaufträge voneinander.



## Selbstsuspendierung

Selbstsuspendierung ermöglicht die erneute Blockierung eines Jobs

👉 Aufgaben, die sich für eine bestimmte Zeit selbst suspendieren verhalten sich **nicht mehr wie periodische Aufgaben** [7, S. 164]

- Beanspruchen in bestimmten Zeitintervallen mehr Rechenzeit  
→ Weitere Verzögerung anderer Arbeitsaufträge
- Zusätzliche Blockierungszeit  $b_i^{ss}$  lässt sich nach oben abschätzen:

$$b_i^{ss} = s_i + \sum_{k=1}^{i-1} \min(e_k, s_k)$$

- Hierbei ist  $s_i$  die längste Selbstsuspendierung von  $T_i$
- Selbstsuspendierung höherpriorer Aufgaben  $T_k$  reduziert deren Einfluss

■ Zusammensetzung der maximale Gesamtdauer  $b_i$

- Aufgabe  $T_i$  suspendiert sich  $K_i$ -mal
- Jedes mal kann sie erneut für  $b_i^{np}$  Zeiteinheiten blockiert werden

**Priority Ceiling**  $b_i = b_i^{ss} + (K_i + 1)b_i^{rc} + (K_i + 1)b_i^{np}$  (vgl. [7, S. 325])



## Erweiterte Planbarkeitsanalyse

Antwortzeit und Auslastung

■ Die Planbarkeitsanalyse einer Aufgabe  $T_i$  unter Berücksichtigung der (gesamten) Blockierungszeit  $b_i$ :

- Bestimmung der Antwortzeit (siehe IV-2/32):

$$\omega_i(t) = e_i + b_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k; 0 < t \leq p_i$$

- Betrachtung der CPU-Auslastung (siehe IV-2/26):

$$\sum_{k=1}^n \frac{e_k}{\min(D_k, p_k)} + \frac{b_i}{\min(D_i, p_i)} \leq 1 \quad ; i = 1, 2, \dots, n$$



## Gliederung

- 1 Überblick
- 2 Konkurrenz und Koordination
  - Kausalordnung und Koordinierung
  - Konkurrenz und Konflikte
- 3 Synchronisation Considered Harmful
- 4 Echtzeitfähige Synchronisationsprotokolle
  - Verdrängungssteuerung
  - Prioritätsvererbung
  - Prioritätsobergrenzen
- 5 Ablaufplanung
- 6 Zusammenfassung



## Zusammenfassung

**Konkurrenz und Koordination** nebenläufiger Aktivitäten

- Nebenläufigkeit, Kausalität, Kausalordnung
- Konfliktsituationen  $\leadsto$  **synchronisieren ohne Prioritätsumkehr**

**Verdrängungssteuerung**  $\rightarrow$  verdrängungsfreie kritische Abschnitte

- benötigt kein *à priori* Wissen; Verklemmungsvorbeugung
- pragmatisch/effektiv, beeinträchtigt unabhängige Jobs

**Prioritätsvererbung**  $\rightarrow$  Priorität zeitweise erhöhen

- benötigt kein *à priori* Wissen
- direkte Blockierung, Blockierung durch Vererbung; transitiv

**Prioritätsobergrenzen**  $\rightarrow$  Priorität zeitweise deckeln

- benötigt *à priori* Wissen; Verklemmungsvorbeugung
- Grundmodell vs. (einfachere) stapelorientierte Variante

**Ablaufplanung**  $\rightarrow$  berücksichtigt Blockierungszeit

- Verzicht auf den Prozessor ermöglicht eine mehrfache Blockierung



## Literaturverzeichnis

- [1] Baker, T. P.:  
A Stack-Based Resource Allocation Policy for Real-Time Processes.  
In: *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS '90)*.  
Lake Buena Vista, FL, USA : IEEE, Dez. 5–7, 1990, S. 191–200
- [2] Baker, T. P.:  
Stack-Based Scheduling of Realtime Processes.  
In: *Real-Time Systems* 3 (1991), Nr. 1, S. 67–99
- [3] Hansen, P. B.:  
*Operating System Principles*.  
Prentice Hall International, 1973
- [4] Hoare, C. A. R.:  
Monitors: An Operating System Structuring Concept.  
In: *Communications of the ACM* 17 (1974), Okt., Nr. 10, S. 549–557
- [5] Jones, M. B.:  
*What really happened on Mars?*  
[http://research.microsoft.com/en-us/um/people/mbj/mars\\_pathfinder/](http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/), 1997
- [6] Lampson, B. W. ; Redell, D. D.:  
Experiences with Processes and Monitors in Mesa.  
In: *Communications of the ACM* 23 (1980), Nr. 2, S. 105–117



## Literaturverzeichnis (Forts.)

- [7] Liu, J. W. S.:  
*Real-Time Systems*.  
Englewood Cliffs, NJ, USA : Prentice Hall PTR, 2000. –  
ISBN 0–13–099651–3
- [8] Mok, A. K.-L. :  
*Fundamental Design Problems of Distributed Systems for Hard Real-Time Environments*.  
Cambridge, MA, USA, Massachusetts Institute of Technology, MIT, Diss., Mai 1983
- [9] Schröder-Preikschat, W. :  
*Softwaresysteme 1*.  
[www4.informatik.uni-erlangen.de/Lehre/SS07/V-S051](http://www4.informatik.uni-erlangen.de/Lehre/SS07/V-S051), 2007. –  
Lecture Notes
- [10] Sha, L. ; Rajkumar, R. ; Lehoczky, J. P.:  
Priority Inheritance Protocols: An Approach to Real-Time Synchronization.  
In: *IEEE Transactions on Computers* 39 (1990), Sept., Nr. 9, S. 1175–1185
- [11] Wilner, D. :  
*Vx-Files: What really happened on Mars?*  
San Francisco, CA, USA : Keynote at the 18th IEEE Real-Time Systems Symposium (RTSS '97), Dez. 1997

