

Echtzeitsysteme

Zugriffskontrolle

Peter Ulbrich

Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

https://www4.cs.fau.de/Lehre/WS19/V_EZS/

13. Januar 2020



☞ Terminvereinbarung zur mündlichen Prüfung

- Erfolgt elektronisch (Poll) → [Windhundverfahren](#)
- Implizite Terminbestätigung → Übertrag in meinCampus

⚠ Anmeldefrist: 13.01.2020 (16 Uhr) bis 02.02.2020

¹Sonst bitte umgehend Mail an uns!

☞ Terminvereinbarung zur mündlichen Prüfung

- Erfolgt elektronisch (Poll) → **Windhundverfahren**
 - Implizite Terminbestätigung → Übertrag in meinCampus
- ⚠ Anmeldefrist:** 13.01.2020 (16 Uhr) bis 02.02.2020

☞ Anmeldung erfolgt in Waffel

- Link zur Teilnahme →
<https://waffel.informatik.uni-erlangen.de/signup?course=406>
und Mail (heute, ca. 14 Uhr) an alle Angemeldeten¹

¹Sonst bitte umgehend Mail an uns!

- ☞ Terminvereinbarung zur mündlichen Prüfung
 - Erfolgt elektronisch (Poll) → **Windhundverfahren**
 - Implizite Terminbestätigung → Übertrag in meinCampus
- ⚠ **Anmeldefrist:** 13.01.2020 (16 Uhr) bis 02.02.2020

- ☞ Anmeldung erfolgt in Waffel
 - Link zur Teilnahme →
<https://waffel.informatik.uni-erlangen.de/signup?course=406>
und Mail (heute, ca. 14 Uhr) an alle Angemeldeten¹

- Terminliche Probleme, Änderungen und Abmeldung
 - Individuelle Terminvereinbarung in Ausnahmefällen möglich
 - Rechtzeitige Abmeldung erlaubt uns Reorganisation
 - Wir beißen nicht!

¹Sonst bitte umgehend Mail an uns!



Evaluation der Veranstaltung

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





Evaluation der Veranstaltung

- Eure Meinung (**Lob/Kritik**) ist uns wichtig!
- Eure Rückmeldung hat Konsequenzen (z.B. Folien-Redesign)
→ Bitte evaluier **Vorlesung** und **Übungen**



Rückläuferquote im Durchschnitt ↪ **10 – 50%**

- Zu wenig für eine sinnvolle Einschätzung





Evaluation der Veranstaltung

- Eure Meinung (**Lob/Kritik**) ist uns wichtig!
- Eure Rückmeldung hat Konsequenzen (z.B. Folien-Redesign)
→ Bitte evaluier **Vorlesung** und **Übungen**



Rückläuferquote im Durchschnitt ↪ **10 – 50%**

- Zu wenig für eine sinnvolle Einschätzung
- Aber: Typische Rückläuferquote in EZS ↪ **70 – 80%**





Evaluation der Veranstaltung

- Eure Meinung (**Lob/Kritik**) ist uns wichtig!
- Eure Rückmeldung hat Konsequenzen (z.B. Folien-Redesign)
→ Bitte evaluier Vorlesung und Übungen



Rückläuferquote im Durchschnitt → **10 – 50%**

- Zu wenig für eine sinnvolle Einschätzung
- Aber: Typische Rückläuferquote in EZS → **70 – 80%**

Motivationsanreiz zur Evaluation



- **Traditionell:** Kaffee und Kekse in der letzten Vorlesung
- **Feste Bedingung:** $\geq 70\%$ der ausgegebenen TANs werden evaluiert!



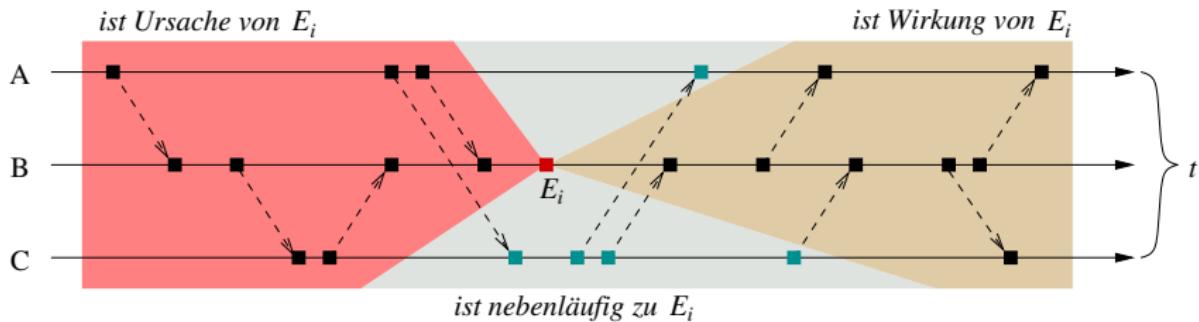
- 1 Überblick**
- 2 Konkurrenz und Koordination**
 - Kausalordnung und Koordinierung
 - Konkurrenz und Konflikte
- 3 Effekte in Echtzeitsystemen**
- 4 Echtzeitfähige Synchronisationsprotokolle**
 - Verdrängungssteuerung
 - Prioritätsvererbung
 - Prioritätsobergrenzen
- 5 Ablaufplanung**
- 6 Zusammenfassung**

- Gegenseitiger Ausschluss in Echtzeitsystemen?
 - Betriebsmittel und Betriebsmittelarten
 - Konkurrenz, Wettstreit 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

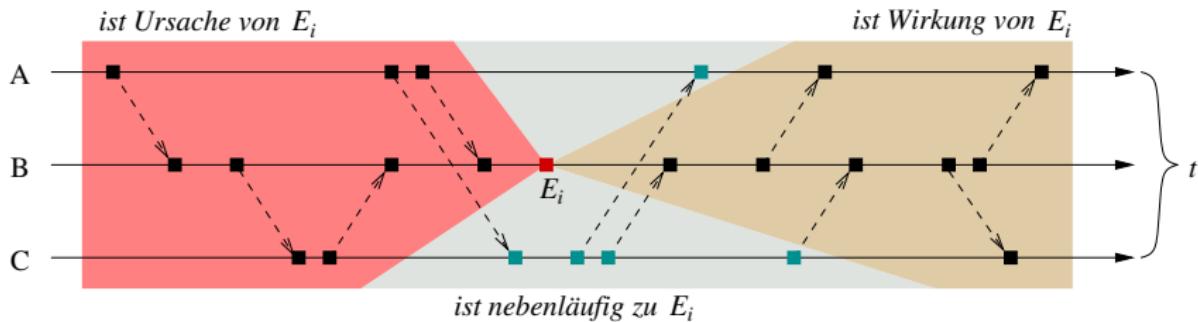


- 1 Überblick
- 2 Konkurrenz und Koordination
 - Kausalordnung und Koordinierung
 - Konkurrenz und Konflikte
- 3 Effekte in Echtzeitsystemen
- 4 Echtzeitfähige Synchronisationsprotokolle
 - Verdrängungssteuerung
 - Prioritätsvererbung
 - Prioritätsobergrenzen
- 5 Ablaufplanung
- 6 Zusammenfassung



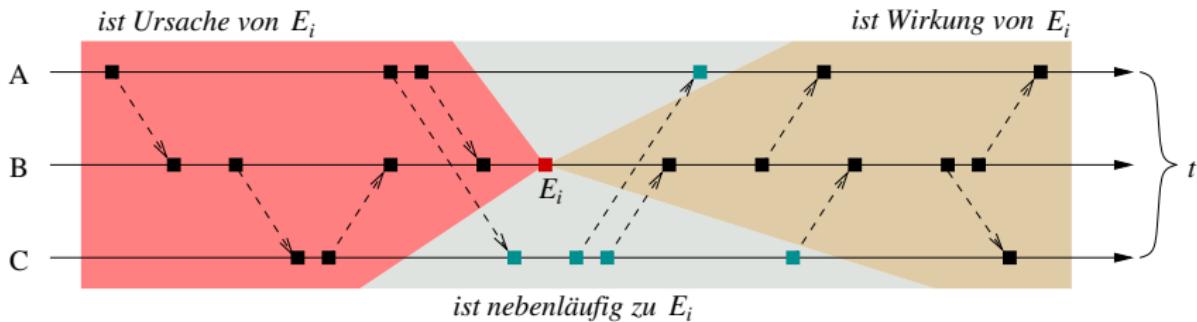


- **Relationen:** Ursache \leftrightarrow Wirkung \leftrightarrow Nebenläufigkeit



- **Relationen:** Ursache \leftrightarrow Wirkung \leftrightarrow Nebenläufigkeit
- ⚠ Ein Ereignis E_i ist nebenläufig zu einem anderen:
→ Es liegt im **Anderswo** anderen Ereignisses





- **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** \mapsto zeitlich geordnete Ereignisse (vgl. Kapitel IV)
 - **Zugriffskontrolle** \mapsto nebenläufige Ereignisse





- **Synchronisation** (gr. *sýn*: zusammen, *chrónos*: Zeit)
 - „Herstellen von Gleichzeitigkeit“





- **Synchronisation** (gr. *sýn*: zusammen, *chrónos*: Zeit)

- „Herstellen von Gleichzeitigkeit“
- Koordination der Kooperation und Konkurrenz zwischen Aufgaben
- Abgleich von Echtzeituhren (oder Daten) in verteilten Systemen





■ **Synchronisation** (gr. *sýn*: 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





■ **Synchronisation** (gr. *sýn*: 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

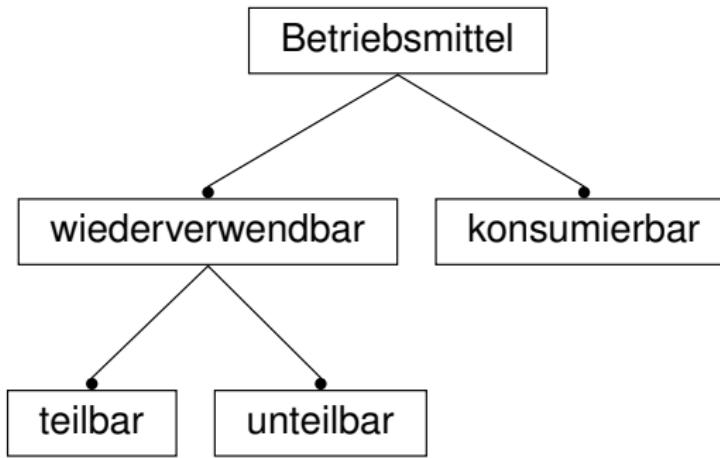
- **Synchronisation** gleichzeitiger Zugriffe auf **gemeinsame Betriebsmittel**
- Herstellen einer Rangfolge für nebenläufige Ereignisse
- Sequentialisierung von Arbeitsaufträgen entlang einer Kausalordnung

⚠ Umsetzung der Zugriffskontrolle (vgl. VI/13 ff)

- **Implizit** in taktgesteuerten Systemen → geeigneter Ablaufplan
- **Explizit** in ereignisgesteuerten Systemen → Synchronisationsprotokolle
- Analytische versus konstruktive Maßnahmen (vgl. VI/13 ff)



Betriebsmittel (engl. *resource*) und Betriebsmittelarten



Hardware

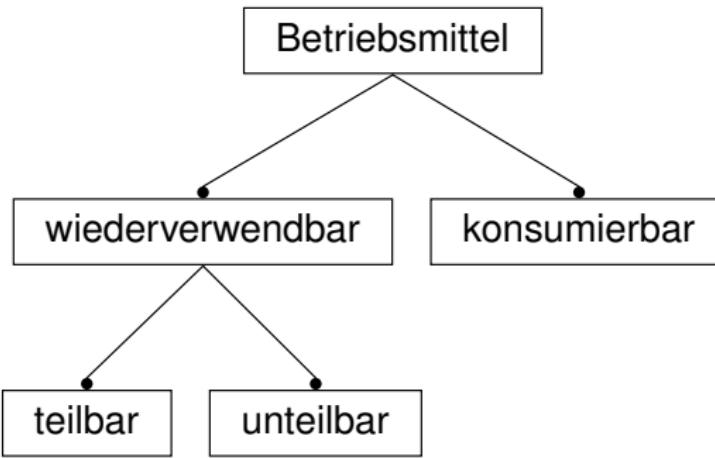
CPU, Speicher, Geräte, Signale

Software

Dateien, Prozesse, Seiten, Puffer, Signale, Nachrichten



Betriebsmittel (engl. *resource*) und Betriebsmittelarten

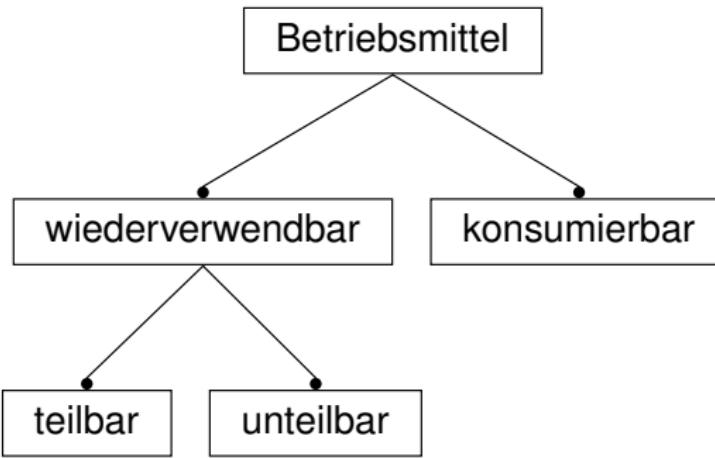


Hardware
CPU, Speicher, Geräte, Signale

Software
Dateien, Prozesse, Seiten, Puffer, Signale, Nachrichten

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

Betriebsmittel (engl. *resource*) und Betriebsmittelarten

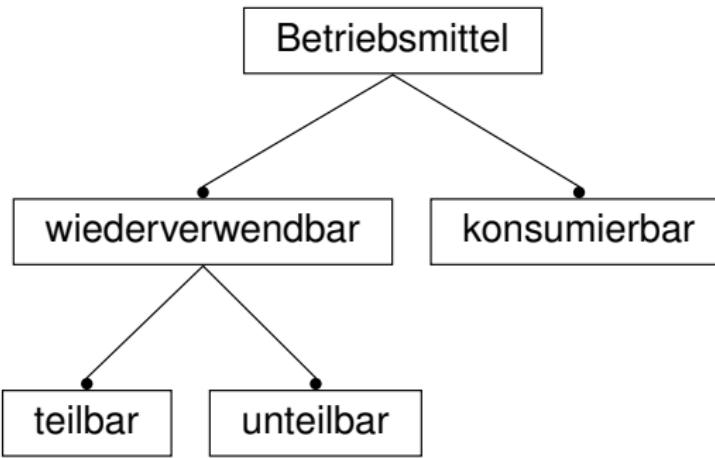


Hardware
CPU, Speicher, Geräte, Signale

Software
Dateien, Prozesse, Seiten, Puffer, Signale, Nachrichten

- **Wettstreit um Betriebsmittel** (engl. *resource contention*) bezieht sich auf Anzahl und Art eines **Betriebsmittels** (engl. *resource*) (R)
einseitige Synchronisation \mapsto konsumierbare Betriebsmittel (\rightarrow Kapitel IV)

Betriebsmittel (engl. *resource*) und Betriebsmittelarten



Hardware
CPU, Speicher, Geräte, Signale

Software
Dateien, Prozesse, Seiten, Puffer, Signale, Nachrichten

- **Wettstreit um Betriebsmittel** (engl. *resource contention*) bezieht sich auf Anzahl und Art eines **Betriebsmittels** (engl. *resource*) (R)
 - einseitige Synchronisation \rightarrow konsumierbare Betriebsmittel (\rightarrow Kapitel IV)
 - mehrseitige Synchronisation \rightarrow wiederverwendbare Betriebsmittel
 - Begrenzung, gegenseitiger Ausschluss





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

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





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

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





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

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

■ **Wettstreit** (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



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



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 gesperrten 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



- 1 Überblick**
- 2 Konkurrenz und Koordination**
 - Kausalordnung und Koordinierung
 - Konkurrenz und Konflikte
- 3 Effekte in Echtzeitsystemen**
- 4 Echtzeitfähige Synchronisationsprotokolle**
 - Verdrängungssteuerung
 - Prioritätsvererbung
 - Prioritätsobergrenzen
- 5 Ablaufplanung**
- 6 Zusammenfassung**





Kritischer Abschnitt (engl. *critical section*), *cs*

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





Kritischer Abschnitt (engl. *critical section*), *cs*

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

■ Beispiel: Blockierung durch Systemaufrufe

- Unterbindung von Verdrängung bei Ausführung von Systemaufrufen





Kritischer Abschnitt (engl. *critical section*), *cs*

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

■ Beispiel: Blockierung durch Systemaufrufe

- Unterbindung von Verdrängung bei Ausführung von Systemaufrufen
- Auswirkungen auf das Laufzeitverhalten von Arbeitsaufträgen:
 - Auftrag J_i läuft und tätigt einen Systemaufruf
 $\leadsto J_i$ hat eine niedrige Priorität, durchläuft unverdrängbar den Kern





Kritischer Abschnitt (engl. *critical section*), *cs*

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





Kritischer Abschnitt (engl. *critical section*), *cs*

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

■ Beispiel: Blockierung durch Systemaufrufe

- Unterbindung von Verdrängung bei Ausführung von Systemaufrufen
- Auswirkungen auf das Laufzeitverhalten von Arbeitsaufträgen:
 - Auftrag J_i läuft und tätigt einen Systemaufruf
 - $\leadsto J_i$ 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_i blockiert bzw. hemmt J_h , die Priorität von J_h wird verletzt





Kritischer Abschnitt (engl. *critical section*), *cs*

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

■ Beispiel: Blockierung durch Systemaufrufe

- Unterbindung von Verdrängung bei Ausführung von Systemaufrufen
- Auswirkungen auf das Laufzeitverhalten von Arbeitsaufträgen:
 - Auftrag J_i läuft und tätigt einen Systemaufruf
 - $\leadsto J_i$ 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_i blockiert bzw. hemmt J_h , die Priorität von J_h wird verletzt



Synchronisation ist **nicht-funktionale Eigenschaft**

(in diesem Fall der Systemaufrufe)



- ☞ **Prioritätsumkehr** [6] ist Folge der Blockierung eines höheren durch einen niedrigeren priorisierten Auftrag

- 👉 **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

- 👉 **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

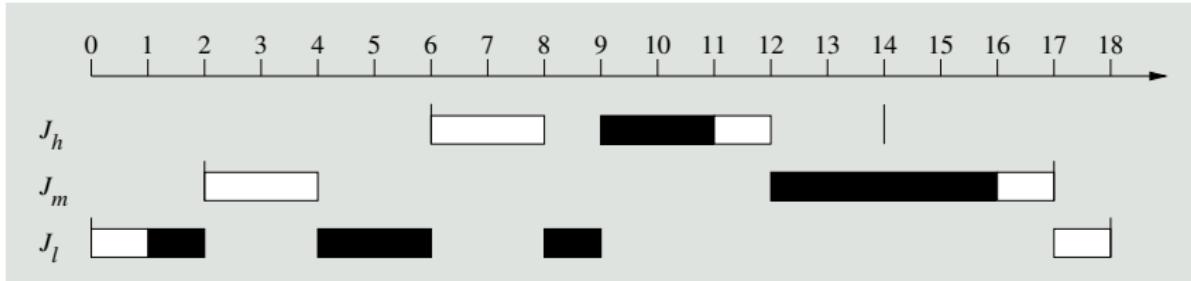
- ☞ **Prioritätsumkehr** [6] ist Folge der Blockierung eines höher durch einen niedriger priorisierten Auftrag
- 1 Der niedrig priorisierte Auftrag durchläuft einen kritischen Abschnitt und wird vom höher priorisierten Auftrag verdrängt
- 2 Der höher priorisierte Auftrag möchte denselben kritischen Abschnitt betreten, wird vom niedrig priorisierten Auftrag jedoch daran gehindert
- 3 Der niedrig priorisierte Job kann weiter ausgeführt werden, obwohl ein höher priorisierter Job wartet

- ☞ **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

$J_l = (0, 6, 18)$, $J_m = (2, 7, 17)$, $J_h = (6, 5, 14)$, alle fordern R an



J_l (niedrige Priorität)

t_0 startet

t_1 belegt Betriebsmittel R

t_4 setzt Ausführung fort

t_8 setzt Ausführung fort

t_9 gibt R frei $\rightarrow J_h$

t_{17} setzt Ausführung fort

t_{18} beendet die Ausführung

J_m (mittlere Priorität)

t_2 startet, verdrängt J_l

t_4 fordert R an, blockiert

t_{12} belegt R

t_{16} gibt R frei

t_{17} beendet die Ausführung

J_h (hohe Priorität)

t_6 startet, verdrängt J_l

t_8 fordert R an, blockiert

t_9 deblockiert, belegt R

t_{11} gibt R frei $\sim J_m$

t_{12} beendet die Ausführung



- **bc_sched** → Task mit höchster Priorität (mit Ausnahme der VxWorks Task „tExec“)
 - Kontrollierte Transaktionen des „1553“-Bus
 - Dieser koppelte Fahr- und Landeeinheit der Raumsonde

- **bc_sched** → Task mit höchster Priorität (mit Ausnahme der VxWorks Task „tExec“)
 - Kontrollierte Transaktionen des „1553“-Bus
 - Dieser koppelte Fahr- und Landeeinheit der Raumsonde

- **bc_dist** → Task mit dritthöchster Priorität
 - Steuerte die Einsammlung der Transaktionsergebnisse
 - Dateneingabe über doppelt gepufferten gemeinsamen Speicher

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

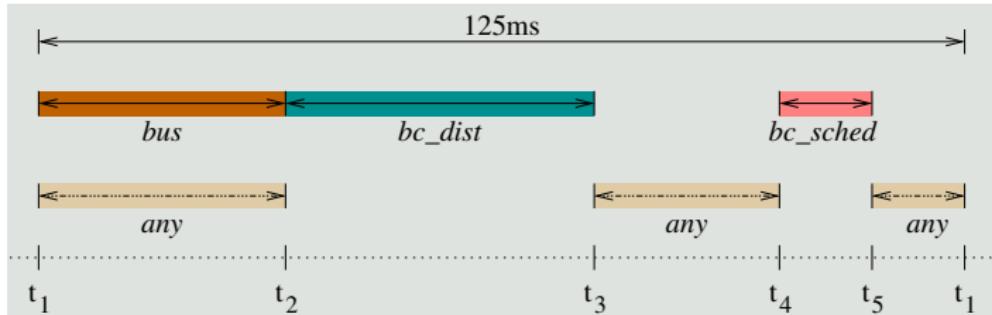


- **bc_sched** → Task mit höchster Priorität (mit Ausnahme der VxWorks Task „tExec“)
 - Kontrollierte Transaktionen des „1553“-Bus
 - Dieser koppelte Fahr- und Landeeinheit der Raumsonde
- **bc_dist** → Task mit dritthöchster Priorität
 - Steuerte die Einsammlung der Transaktionsergebnisse
 - Dateneingabe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

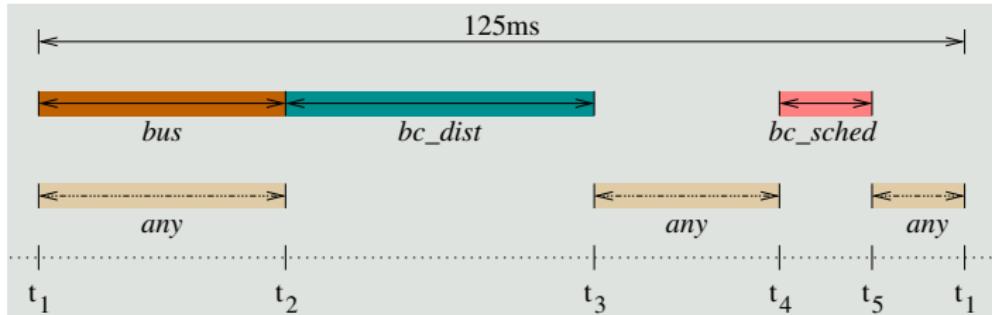


Hardware gab eine Periodenlänge von 8 Hz (d.h., 125 ms) vor





- 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



- 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





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





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





- ASI/MET (niedrige Priorität) hat bc_dist (hohe Priorität) blockiert:
 - ASI/MET belegte 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

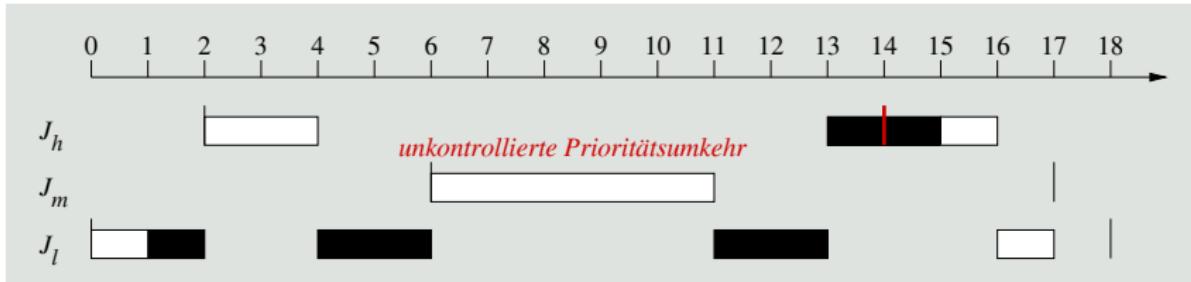


- ☞ ASI/MET (niedrige Priorität) hat bc_dist (hohe Priorität) blockiert:
 - ASI/MET belegte wiederverwendbares, unteilbares Betriebsmittel
 - Wurde von bc_dist angefordert, bevor ASI/MET es wieder frei gab
 - Im weiteren Verlauf verdrängten Tasks mittlerer Priorität ASI/MET
 - Verlängerung der **Blockierungszeit** 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: $J_l = (0, 7, 18)$, $J_m = (6, 5, 17)$, $J_h = (2, 5, 14)$, J_m fordert R nicht an



J_l (niedrige Priorität)

- t_0 startet
- t_1 belegt Betriebsmittel R
- t_4 setzt Ausführung fort
- t_{11} setzt Ausführung fort
- t_{13} gibt R frei $\rightarrow J_h$
- t_{16} setzt Ausführung fort
- t_{17} beendet die Ausführung

J_m (mittlere Priorität)

- t_6 startet, verdrängt J_l
- t_{11} beendet die Ausführung

J_h (hohe Priorität)

- t_2 startet, verdrängt J_l
- t_4 fordert R an, blockiert
- t_{13} belegt R
- t_{14} verletzt seinen Termin
- t_{15} gibt R frei
- t_{16} beendet die Ausführung





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





- Prioritätsumkehr (siehe Folie 14) ist ein mögliches Phänomen abhängiger Aufträge, welches in zwei Ausprägungen auftreten kann:
 - 1 (Normale) Prioritätsumkehr (engl. *priority inversion*)
 - Gegenseitiger Ausschluss → Ein Auftrag hoher Priorität wartet auf Einen niedriger Priorität (ggf. unvermeidbar)





- Prioritätsumkehr (siehe Folie 14) ist ein mögliches Phänomen abhängiger Aufträge, welches in zwei Ausprägungen auftreten kann:
 - 1 (Normale) Prioritätsumkehr (engl. *priority inversion*)
 - Gegenseitiger Ausschluss → Ein Auftrag hoher Priorität wartet auf Einen niedriger Priorität (ggf. unvermeidbar)
 - 2 Unkontrollierte Prioritätsumkehr (engl. *unbounded priority inversion*)
 - Der Auftrag niedriger Priorität wird von unbeteiligten Aufträge mittlerer Priorität verdrängt





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

- 1 (Normale) Prioritätsumkehr (engl. *priority inversion*)
 - Gegenseitiger Ausschluss → Ein Auftrag hoher Priorität wartet auf Einen niedriger Priorität (ggf. unvermeidbar)
- 2 Unkontrollierte Prioritätsumkehr (engl. *unbounded priority inversion*)
 - Der Auftrag niedriger Priorität wird von unbeteiligten Aufträge mittlerer Priorität verdrängt



Lösungsansätze für die blockierende Synchronisation sind:

- Verdrängungssteuerung
- Prioritätsvererbung
- Prioritätsobergrenzen

- Folie 23 ff.
- Folie 28 ff.
- Folie 32 ff.



- 1 Überblick
- 2 Konkurrenz und Koordination
 - Kausalordnung und Koordinierung
 - Konkurrenz und Konflikte
- 3 Effekte in Echtzeitsystemen
- 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, ...





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** → Verdrängung abwehren
 - Ausgelöste Aufträge einplanen, aber nicht einlasten





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** → Verdrängung abwehren
 - Ausgelöste Aufträge einplanen, aber nicht einlasten
 - Austrittsprotokoll** → Verdrängung wieder zulassen
 - Höher priorisierte Aufträge (nachträglich) einlasten





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 → Verdrängung abwehren

- Ausgelöste Aufträge einplanen, aber nicht einlasten

Austrittsprotokoll → Verdrängung wieder zulassen

- Höher priorisierte Aufträge (nachträglich) einlasten



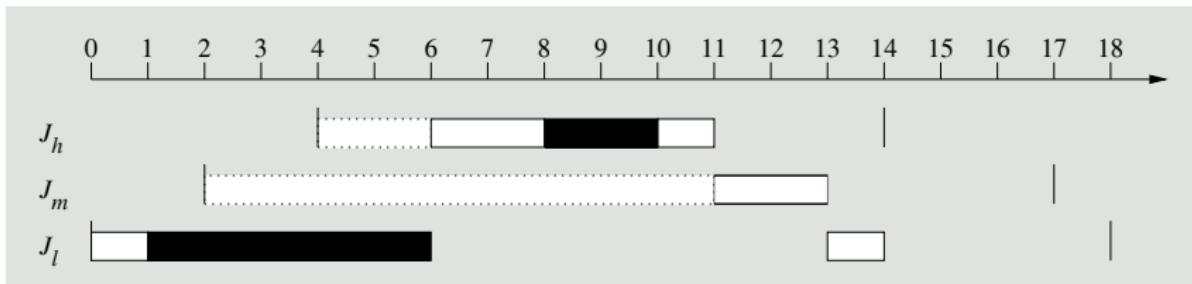
Aufträge **unverdrängbar** bis zur Freigabe des Betriebsmittels

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



Beispiel: Verdrängungssteuerung

Beispiel: $J_l = (0, 7, 18)$, $J_m = (6, 5, 17)$, $J_h = (2, 5, 14)$, J_m fordert R nicht an



J_l (niedrige Priorität)

t_0 startet

t_1 belegt R unverdrängbar

t_6 gibt R frei $\mapsto J_h$

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

t_{10} gibt R frei

t_{11} 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





Verzögerungen nebenläufiger Arbeitsaufträge durch die Zugriffskontrolle sind nach oben begrenzt





- Verzögerungen nebenläufiger Arbeitsaufträge durch die Zugriffskontrolle sind nach oben begrenzt
 - Höher priorisierte Aufträge werden schlimmstenfalls **einmal** durch einen niedriger priorisierten Auftrag blockiert





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^c (*resource contention*) bestimmt sich aus der größten WCET aller kritischen Abschnitte aller niedriger priorisierten Aufträge: $\max(cs)$





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_i^{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





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 $D_i < D_j$



- ☞ Kein à priori Wissen über Betriebsmittelanforderungen notwendig

☞ 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



☞ 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

☞ 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





Benachteiligt an Betriebsmitteln unbeteiligte Arbeitsaufträge

- Blockierung hochpriorer Aufträge ohne Konfliktsituation
- Im Beispiel (vgl. Folie 24) wird J_m durch J_l blockiert, obwohl beide Aufträge nicht im gegenseitigen Ausschluss zueinander stehen



Benachteiligt an Betriebsmitteln unbeteiligte Arbeitsaufträge

- Blockierung hochpriorer Aufträge ohne Konfliktsituation
- Im Beispiel (vgl. Folie 24) wird J_m durch J_l blockiert, obwohl beide Aufträge nicht im gegenseitigen Ausschluss zueinander stehen



Verbesserungsmöglichkeiten

- 1 Durch 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





☞ 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





☞ 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 niederpriore 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





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 niederpriore 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



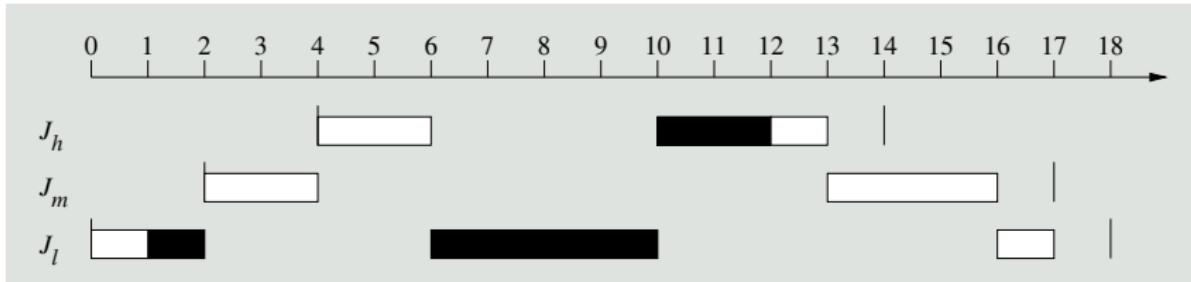
Prioritätsumkehr wird nicht vermieden, jedoch entschärft:

- Verdrängbarkeit durch unbeteiligte Arbeitsaufträge höherer Priorität



Beispiel: Prioritätsvererbung

Beispiel: $J_l = (0, 7, 18)$, $J_m = (2, 7, 17)$, $J_h = (4, 5, 14)$, J_m fordert R nicht an



J_l (niedrige Priorität)

t_0 startet

t_1 belegt R

t_2 Verdrängung durch J_m

t_6 erbt die Priorität von J_h

t_{10} gibt R frei $\rightarrow J_h$

t_{16} läuft mit alter Priorität

t_{17} beendet die Ausführung

J_m (mittlere Priorität)

t_2 startet

t_4 Verdrängung durch J_h

t_{13} läuft weiter

t_{16} beendet die Ausführung

J_h (hohe Priorität)

t_4 startet

t_6 fordert R an \rightarrow Prioritätsvererbung

t_{10} belegt R

t_{12} gibt R frei

t_{13} beendet die Ausführung



Prioritätsvererbung bedingt **zwei Arten** von Blockierung





Prioritätsvererbung bedingt **zwei Arten** von Blockierung

1

Direkte Blockierung (engl. *direct blocking*)

- Bekannte Blockierung eines höher priorisierten Auftrags (J_h) durch einen niedriger priorisierter Auftrag (J_l), welcher das angeforderte Betriebsmittel hält





Prioritätsvererbung bedingt **zwei Arten** von Blockierung

1 Direkte Blockierung (engl. *direct blocking*)

- Bekannte Blockierung eines höher priorisierten Auftrags (J_h) durch einen niedriger priorisierten Auftrag (J_l), welcher das angeforderte Betriebsmittel hält

2 Blockierung durch Vererbung (engl. *inheritance blocking*)

- Blockierung eines **nicht im gegenseitigen Ausschluss befindlichen** höher priorisierten Auftrags (J_m)
- Welcher J_l gemäß dessen „Altpriorität“ verdrängen könnte, dies jedoch wegen der geerbten Priorität nicht kann





Prioritätsvererbung bedingt **zwei Arten** von Blockierung

1 Direkte Blockierung (engl. *direct blocking*)

- Bekannte Blockierung eines höher priorisierten Auftrags (J_h) durch einen niedriger priorisierten Auftrag (J_l), welcher das angeforderte Betriebsmittel hält

2 Blockierung durch Vererbung (engl. *inheritance blocking*)

- Blockierung eines **nicht im gegenseitigen Ausschluss** befindlichen höher priorisierten Auftrags (J_m)
- Welcher J_l gemäß dessen „Altpriorität“ verdrängen könnte, dies jedoch wegen der geerbten Priorität nicht kann



Transitive Blockierung bei geschachtelten kritischen Abschnitten





Prioritätsvererbung bedingt **zwei Arten** von Blockierung

1 Direkte Blockierung (engl. *direct blocking*)

- Bekannte Blockierung eines höher priorisierten Auftrags (J_h) durch einen niedriger priorisierten Auftrag (J_l), welcher das angeforderte Betriebsmittel hält

2 Blockierung durch Vererbung (engl. *inheritance blocking*)

- Blockierung eines **nicht im gegenseitigen Ausschluss** befindlichen höher priorisierten Auftrags (J_m)
- Welcher J_l gemäß dessen „Altpriorität“ verdrängen könnte, dies jedoch wegen der geerbten Priorität nicht kann



Transitive Blockierung bei geschachtelten kritischen Abschnitten

1 J_l startet zuerst, belegt R_1 und wird von J_m verdrängt





☞ Prioritätsvererbung bedingt **zwei Arten** von Blockierung

1 Direkte Blockierung (engl. *direct blocking*)

- Bekannte Blockierung eines höher priorisierten Auftrags (J_h) durch einen niedriger priorisierten Auftrag (J_l), welcher das angeforderte Betriebsmittel hält

2 Blockierung durch Vererbung (engl. *inheritance blocking*)

- Blockierung eines **nicht im gegenseitigen Ausschluss** befindlichen höher priorisierten Auftrags (J_m)
- Welcher J_l gemäß dessen „Altpriorität“ verdrängen könnte, dies jedoch wegen der geerbten Priorität nicht kann



Transitive Blockierung bei geschachtelten kritischen Abschnitten

- 1 J_l startet zuerst, belegt R_1 und wird von J_m verdrängt
- 2 J_m belegt R_2 und wird von J_h verdrängt





☞ Prioritätsvererbung bedingt **zwei Arten** von Blockierung

1 Direkte Blockierung (engl. *direct blocking*)

- Bekannte Blockierung eines höher priorisierten Auftrags (J_h) durch einen niedriger priorisierten Auftrag (J_l), welcher das angeforderte Betriebsmittel hält

2 Blockierung durch Vererbung (engl. *inheritance blocking*)

- Blockierung eines **nicht im gegenseitigen Ausschluss** befindlichen höher priorisierten Auftrags (J_m)
- Welcher J_l gemäß dessen „Altpriorität“ verdrängen könnte, dies jedoch wegen der geerbten Priorität nicht kann



Transitive Blockierung bei geschachtelten kritischen Abschnitten

- 1 J_l startet zuerst, belegt R_1 und wird von J_m verdrängt
- 2 J_m belegt R_2 und wird von J_h verdrängt
- 3 J_h fordert R_2 an und vererbt seine Priorität an J_m





☞ Prioritätsvererbung bedingt **zwei Arten** von Blockierung

1 Direkte Blockierung (engl. *direct blocking*)

- Bekannte Blockierung eines höher priorisierten Auftrags (J_h) durch einen niedriger priorisierten Auftrag (J_l), welcher das angeforderte Betriebsmittel hält

2 Blockierung durch Vererbung (engl. *inheritance blocking*)

- Blockierung eines **nicht im gegenseitigen Ausschluss** befindlichen höher priorisierten Auftrags (J_m)
- Welcher J_l gemäß dessen „Altpriorität“ verdrängen könnte, dies jedoch wegen der geerbten Priorität nicht kann



Transitive Blockierung bei geschachtelten kritischen Abschnitten

- 1 J_l startet zuerst, belegt R_1 und wird von J_m verdrängt
- 2 J_m belegt R_2 und wird von J_h verdrängt
- 3 J_h fordert R_2 an und vererbt seine Priorität an J_m
- 4 J_m läuft weiter, fordert R_1 an und vererbt „seine“ Priorität an J_l



Prioritätsvererbung: Blockierungszeit

Feste obere Schranken, die kaskadenartig zur Wirkung kommen können



- Schlimmster Fall:** Auftrag J_i benötigt $n \geq 1$ Betriebsmittel und steht mit $k \geq 1$ niedriger priorisierten Aufträgen im Konflikt
- J_i kann $\min(n, k)$ -mal blockiert werden
 - Für die Dauer der WCET des äußersten kritischen Abschnitts



Prioritätsvererbung: Blockierungszeit

Feste obere Schranken, die kaskadenartig zur Wirkung kommen können

- ☞ **Schlimmster Fall:** Auftrag J_i benötigt $n \geq 1$ Betriebsmittel und steht mit $k \geq 1$ niedriger priorisierten Aufträgen im Konflikt
 - J_i kann $\min(n, k)$ -mal blockiert werden
 - Für die Dauer der WCET des äußersten kritischen Abschnitts
- Blockierungszeit ist maximal $b_i^{rc} = \min(n, k) \cdot \max_j (cs_j)$
 - cs_j sind kritische Abschnitte von Aufträgen niedrigerer Priorität
 - **Pessimistisch:** Kritische Abschnitte sind unterschiedlich lang



Prioritätsvererbung: Blockierungszeit

Feste obere Schranken, die kaskadenartig zur Wirkung kommen können

- ☞ **Schlimmster Fall:** Auftrag J_i benötigt $n \geq 1$ Betriebsmittel und steht mit $k \geq 1$ niedriger priorisierten Aufträgen im Konflikt
 - J_i kann $\min(n, k)$ -mal blockiert werden
 - Für die Dauer der WCET des äußersten kritischen Abschnitts
 - Blockierungszeit ist maximal $b_i^{rc} = \min(n, k) \cdot \max_j (cs_j)$
 - cs_j sind kritische Abschnitte von Aufträgen niedrigerer Priorität
 - **Pessimistisch:** Kritische Abschnitte sind unterschiedlich lang
- ⚠ Der GAU (J_0 höchste Priorität, vgl. [7, S. 289])
- 1 J_k startet zuerst, belegt R_n



Prioritätsvererbung: Blockierungszeit

Feste obere Schranken, die kaskadenartig zur Wirkung kommen können

- ☞ **Schlimmster Fall:** Auftrag J_i benötigt $n \geq 1$ Betriebsmittel und steht mit $k \geq 1$ niedriger priorisierten Aufträgen im Konflikt
 - J_i kann $\min(n, k)$ -mal blockiert werden
 - Für die Dauer der WCET des äußersten kritischen Abschnitts
- Blockierungszeit ist maximal $b_i^{rc} = \min(n, k) \cdot \max_j (cs_j)$
 - cs_j sind kritische Abschnitte von Aufträgen niedrigerer Priorität
 - **Pessimistisch:** Kritische Abschnitte sind unterschiedlich lang
- ⚠ Der GAU (J_0 höchste Priorität, vgl. [7, S. 289])
 - 1 J_k startet zuerst, belegt R_n ; J_{k-1} verdrängt J_k , belegt R_{n-1}



Prioritätsvererbung: Blockierungszeit

Feste obere Schranken, die kaskadenartig zur Wirkung kommen können

- ☞ **Schlimmster Fall:** Auftrag J_i benötigt $n \geq 1$ Betriebsmittel und steht mit $k \geq 1$ niedriger priorisierten Aufträgen im Konflikt
 - J_i kann $\min(n, k)$ -mal blockiert werden
 - Für die Dauer der WCET des äußersten kritischen Abschnitts
 - Blockierungszeit ist maximal $b_i^{rc} = \min(n, k) \cdot \max_j (cs_j)$
 - cs_j sind kritische Abschnitte von Aufträgen niedrigerer Priorität
 - **Pessimistisch:** Kritische Abschnitte sind unterschiedlich lang
- ⚠ Der GAU (J_0 höchste Priorität, vgl. [7, S. 289])
- 1 J_k startet zuerst, belegt R_n ; J_{k-1} verdrängt J_k , belegt R_{n-1} ; ...
 - 2 ... J_1 verdrängt J_2 , belegt R_1



Prioritätsvererbung: Blockierungszeit

Feste obere Schranken, die kaskadenartig zur Wirkung kommen können



- Schlimmster Fall:** Auftrag J_i benötigt $n \geq 1$ Betriebsmittel und steht mit $k \geq 1$ niedriger priorisierten Aufträgen im Konflikt
- J_i kann $\min(n, k)$ -mal blockiert werden
 - Für die Dauer der WCET des äußersten kritischen Abschnitts

- Blockierungszeit ist maximal $b_i^{rc} = \min(n, k) \cdot \max_j (cs_j)$
 - cs_j sind kritische Abschnitte von Aufträgen niedrigerer Priorität
 - **Pessimistisch:** Kritische Abschnitte sind unterschiedlich lang



Der GAU (J_0 höchste Priorität, vgl. [7, S. 289])

- 1 J_k startet zuerst, belegt R_n ; J_{k-1} verdrängt J_k , belegt R_{n-1} ; ...
- 2 ... J_1 verdrängt J_2 , belegt R_1
- 3 J_0 verdrängt J_1 , fordert R_i an in der Reihenfolge $i = 1, 2, 3, \dots, n$





- ☞ **Prioritätsobergrenze** (engl. *priority ceiling*) $\hat{\Pi}$ eines Betriebsmittels R_i ist die höchste Priorität aller beteiligten Arbeitsaufträge
 - Die jeweiligen Werte $\hat{\Pi}$ sind für alle Betriebsmittel **im Voraus** bekannt





- ☞ **Prioritätsobergrenze** (engl. *priority ceiling*) $\hat{\Pi}$ eines Betriebsmittels R_i ist die höchste Priorität aller beteiligten Arbeitsaufträge
 - Die jeweiligen Werte $\hat{\Pi}$ sind für alle Betriebsmittel **im Voraus** bekannt
- **Aktuelle Prioritätsobergrenze** des Systems $\hat{\Pi}(t)$
 - Entspricht der höchsten Obergrenze aller belegten Betriebsmittel
 - ⚠ In Abhängigkeit vom betrachteten Zeitpunkt t





- ☞ **Prioritätsobergrenze** (engl. *priority ceiling*) $\hat{\Pi}$ eines Betriebsmittels R_i ist die höchste Priorität aller beteiligten Arbeitsaufträge
 - Die jeweiligen Werte $\hat{\Pi}$ sind für alle Betriebsmittel **im Voraus** bekannt
- **Aktuelle Prioritätsobergrenze** des Systems $\hat{\Pi}(t)$
 - Entspricht der höchsten Obergrenze aller belegten Betriebsmittel
 - ⚠ In Abhängigkeit vom betrachteten Zeitpunkt t
 - Ist kein Betriebsmittel belegt, existiert die aktuelle Prioritätsobergrenze (theoretisch) nicht
 - $\hat{\Pi}(t)$ ist dann niedriger als die niedrigste Priorität aller Aufträge





- ☞ **Prioritätsobergrenze** (engl. *priority ceiling*) $\hat{\Pi}$ eines Betriebsmittels R_i ist die höchste Priorität aller beteiligten Arbeitsaufträge
 - Die jeweiligen Werte $\hat{\Pi}$ sind für alle Betriebsmittel **im Voraus** bekannt

- **Aktuelle Prioritätsobergrenze** des Systems $\hat{\Pi}(t)$
 - Entspricht der höchsten Obergrenze aller belegten Betriebsmittel
 - ⚠ In Abhängigkeit vom betrachteten Zeitpunkt t
 - Ist kein Betriebsmittel belegt, existiert die aktuelle Prioritätsobergrenze (theoretisch) nicht
 - $\hat{\Pi}(t)$ ist dann niedriger als die niedrigste Priorität aller Aufträge



Prioritätsobergrenzen sind eine **Variante von Prioritätsvererbung**

- Im Konfliktfall erben Arbeitsaufträge die Priorität des anfordernden Auftrags





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



- ☞ 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





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...





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 $P(t) > \hat{\Pi}(t)$





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 $P(t) > \hat{\Pi}(t)$

2 $P(t) \leq \hat{\Pi}(t)$, aber J hält zum Zeitpunkt t mindestens ein Betriebsmittel mit Prioritätsobergrenze $\hat{\Pi}(t)$





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 $P(t) > \hat{\Pi}(t)$

2 $P(t) \leq \hat{\Pi}(t)$, aber J hält zum Zeitpunkt t mindestens ein Betriebsmittel mit Prioritätsobergrenze $\hat{\Pi}(t)$

\rightsquigarrow Andernfalls bleibt R frei und J blockiert (siehe Folie 35)



- ☞ 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...

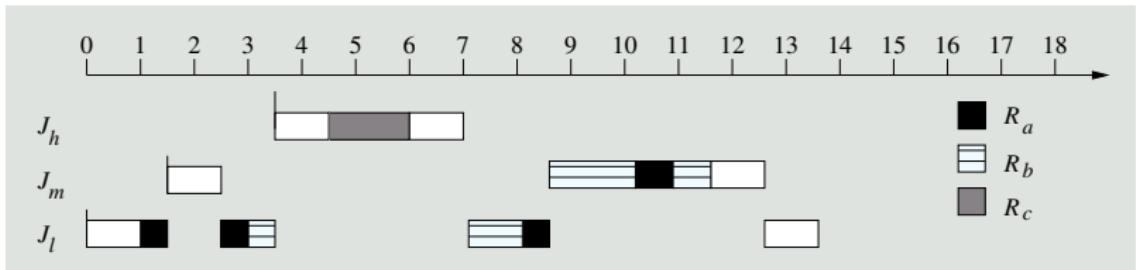
- 1 $P(t) > \hat{\Pi}(t)$
 - 2 $P(t) \leq \hat{\Pi}(t)$, aber J hält zum Zeitpunkt t mindestens ein Betriebsmittel mit Prioritätsobergrenze $\hat{\Pi}(t)$
- ~ Andernfalls bleibt R frei und J blockiert (siehe Folie 35)

- ☞ 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: $J_l = (0, 3.5, 18)$, $J_m = (1.4, 5, 17)$, $J_h = (3.5, 5, 14)$; J_l , J_m teilen sich R_A , R_B



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





- Betriebsmittelvergabe durch Prioritätsobergrenzen ist weniger **gefährig** (engl. *greedy*) als Prioritätsvererbung²
 - Anforderung von J kann zurückgewiesen werden, obwohl das angeforderte Betriebsmittel R frei ist

²Erinnerung: Verklemmungen werden durch Prioritätsvererbung nicht verhindert.



Betriebsmittelvergabe durch Prioritätsobergrenzen ist weniger **gefährlich** (engl. *greedy*) als Prioritätsvererbung²

- 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 $\hat{\Pi}(t)$
 - Die direkte/indirekte Priorität von J durchbricht die Ordnung
- Alle Betriebsmittel des Systems sind linear geordnet aufgestellt

²Erinnerung: Verklemmungen werden durch Prioritätsvererbung nicht verhindert.

- ☞ Betriebsmittelvergabe durch Prioritätsobergrenzen ist weniger **gefährlich** (engl. *greedy*) als Prioritätsvererbung²
 - 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 $\hat{\Pi}(t)$
 - 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

²Erinnerung: Verklemmungen werden durch Prioritätsvererbung nicht verhindert.



Vereinfachung durch Stapelorientierung

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

- Ausgangspunkt ist die **stapelorientierte Einplanung von Prozessen** [1, 2]





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





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





- 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
 - *run-to-completion-Semantik*
 - Oben auf dem Stapel läuft immer der Auftrag mit der höchsten Priorität
 - Logische Konsequenz bei ausgeschlossener Selbstaussetzung





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
 - *run-to-completion-Semantik*
 - 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*)



- Aktualisierung der Priorität $P(t)$ (siehe Folie 32)
 - Erfolgt mit jeder Vergabe/Freigabe von Betriebsmitteln
→ Entspricht einer vorauseilenden Vererbung



- 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



- 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



- 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)



- ☞ Verklemmungen sind durch eine **indirekte Methode** zur **Verklemmungsvorbeugung**³ ausgeschlossen

³ **zirkuläres Warten** wird vorgebeugt (siehe auch Folie 35).

- ☞ Verklemmungen sind durch eine **indirekte Methode** zur **Verklemmungsvorbeugung**³ 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

³ **zirkulares Warten** wird vorgebeugt (siehe auch Folie 35).

- ☞ Verklemmungen sind durch eine **indirekte Methode** zur **Verklemmungsvorbeugung**³ 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

³ **zirkulares Warten** wird vorgebeugt (siehe auch Folie 35).



- ☞ Verklemmungen sind durch eine **indirekte Methode** zur **Verklemmungsvorbeugung**³ 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

³ **zirkuläres Warten** wird vorgebeugt (siehe auch Folie 35).



- ☞ PCP mit festen Prioritäten vergleichsweise einfach



- 👉 PCP mit festen Prioritäten vergleichsweise einfach
- ⚠️ Dynamische Priorität → Prioritäten der Aufgaben ändern sich
→ Grenzprioritäten der Betriebsmittel ändern sich mit der Zeit



- 👉 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



☞ 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



- 👉 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



- 👉 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



- 👉 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





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

- 1 Direkte Blockierung,
- 2 Blockierung durch Vererbung,
- 3 Blockierung durch Aufhebungssperre





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

- 1 Direkte Blockierung,
- 2 Blockierung durch Vererbung,
- 3 Blockierung durch Aufhebungssperre

} Prioritätsvererbung



- ☞ Zugriffskontrolle durch Prioritätsobergrenzen impliziert nunmehr **drei Arten der Blockierung**:
 - 1 Direkte Blockierung,
 - 2 Blockierung durch Vererbung,
 - 3 Blockierung durch Aufhebungssperre
- Effekt von **3.** ist, dass jeder Arbeitsauftrag höchstens einmal blockiert und dass eine Blockierung nicht transitiv ist [10]

} Prioritätsvererbung

- ☞ Zugriffskontrolle durch Prioritätsobergrenzen impliziert nunmehr **drei Arten der Blockierung**:
 - 1 Direkte Blockierung,
 - 2 Blockierung durch Vererbung,
 - 3 Blockierung durch Aufhebungssperre
- 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

- ☞ 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

- ☞ 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 niederiore 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



- 1 Überblick
- 2 Konkurrenz und Koordination
 - Kausalordnung und Koordinierung
 - Konkurrenz und Konflikte
- 3 Effekte in Echtzeitsystemen
- 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 unabhängig⁴

A6 Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar

A7 Alle Aufgaben verhalten sich voll-präemptiv

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



Fadensynchronisation \rightsquigarrow Blockierungszeit

Die Blockierungszeit verzögert die Fertigstellung von Arbeitsaufträgen

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

NPCS (s. Folie 25) $b_i^{rc} = \max_{i+1 \leq k \leq n} (cs_k)$

Priority Inheritance (s. Folie 31) $b_i^{rc} = \min(n, k) \cdot \max_{i+1 \leq l \leq n} (cs_l)$

Priority Ceiling (s. Folie 40) $b_i^{rc} = \max_{i+1 \leq k \leq n} (cs_k)$



Fadensynchronisation \rightsquigarrow Blockierungszeit

Die Blockierungszeit verzögert die Fertigstellung von Arbeitsaufträgen

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

$$\text{NPCS (s. Folie 25)} \quad b_i^{rc} = \max_{i+1 \leq k \leq n} (cs_k)$$

$$\text{Priority Inheritance (s. Folie 31)} \quad b_i^{rc} = \min(n, k) \cdot \max_{i+1 \leq l \leq n} (cs_l)$$

$$\text{Priority Ceiling (s. Folie 40)} \quad b_i^{rc} = \max_{i+1 \leq k \leq n} (cs_k)$$

- 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

$$\text{NPCS} \quad b_i = \max(b_i^{rc}, b_i^{np})$$

$$\text{Priority Inheritance} \quad b_i = \min(n, k) \cdot (b_i^{rc} + b_i^{np})$$

$$\text{Priority Ceiling} \quad b_i = b_i^{rc} + b_i^{np}$$



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 unabhängig⁵
- A6** Die Kosten durch Unterbrechungen, Ablaufplanung und Verdrängung sind vernachlässigbar
- A7** Alle Aufgaben verhalten sich voll-präemptiv

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





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])



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



- Die Planbarkeitsanalyse einer Aufgabe T_i unter Berücksichtigung der (gesamten) Blockierungszeit b_i :
 - Bestimmung der Antwortzeit (siehe IV-2/39):

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



- Die Planbarkeitsanalyse einer Aufgabe T_i unter Berücksichtigung der (gesamten) Blockierungszeit b_i :
 - Bestimmung der Antwortzeit (siehe IV-2/39):

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

- Betrachtung der CPU-Auslastung (siehe IV-2/30):

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



- 1 Überblick
- 2 Konkurrenz und Koordination
 - Kausalordnung und Koordinierung
 - Konkurrenz und Konflikte
- 3 Effekte in Echtzeitsystemen
- 4 Echtzeitfähige Synchronisationsprotokolle
 - Verdrängungssteuerung
 - Prioritätsvererbung
 - Prioritätsobergrenzen
- 5 Ablaufplanung
- 6 Zusammenfassung



Konkurrenz und Koordination nebenläufiger Aktivitäten

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



Konkurrenz und Koordination nebenläufiger Aktivitäten

- Nebenläufigkeit, Kausalität, Kausalordnung
- Konfliktsituationen ↼ **synchronisieren ohne Prioritätsumkehr**

Verdrängungssteuerung ↼ verdrängungsfreie kritische Abschnitte

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

Konkurrenz und Koordination nebenläufiger Aktivitäten

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

Verdrängungssteuerung \leadsto verdrängungsfreie kritische Abschnitte

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

Prioritätsvererbung \leadsto Priorität zeitweise erhöhen

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



Zusammenfassung

Konkurrenz und Koordination nebenläufiger Aktivitäten

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

Verdrängungssteuerung \leadsto verdrängungsfreie kritische Abschnitte

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

Prioritätsvererbung \leadsto Priorität zeitweise erhöhen

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

Prioritätsobergrenzen \leadsto Priorität zeitweise deckeln

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



Konkurrenz und Koordination nebenläufiger Aktivitäten

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

Verdrängungssteuerung \leadsto verdrängungsfreie kritische Abschnitte

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

Prioritätsvererbung \leadsto Priorität zeitweise erhöhen

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

Prioritätsobergrenzen \leadsto Priorität zeitweise deckeln

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

Ablaufplanung \leadsto berücksichtigt Blockierungszeit

- Verzicht auf den Prozessor ermöglicht eine mehrfache Blockierung



- [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/, 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

- [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. –
Technical Report MIT/LCS/TR-297
- [9] Schröder-Preikschat, W. :
Softwaresysteme 1.
www4.informatik.uni-erlangen.de/Lehre/SS07/V_SOS1, 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 : Keynote at the 18th IEEE Real-Time Systems Symposium (RTSS '97), Dez. 1997

EZS – Cheat Sheet

Typographische Konvention

Der erste Index gibt die Aufgabe an (z.B. D_i), der Zweite (optional) bezieht sich auf den Arbeitsauftrag (z.B. $d_{i,j}$). Exponenten zeigen verschiedene Varianten einer Eigenschaft an (z.B. T^{HI}, T^{MED}, T^{LO}). Funktionen beschreiben zeitlich variierende Eigenschaften (z.B. $P(t)$).

Eigenschaften

t (Real-)Zeit

d Zeitverzögerung (engl. delay)

Strukturelemente

E_i Ereignis (engl. event)

R_i Ergebnis (engl. result)

T_i Aufgabe (engl. task)

$J_{i,j}$ Arbeitsauftrag (engl. job) der Aufgabe T_i

Temporale Eigenschaften

Allgemein

r_i Auslösezeitpunkt
(engl. release time)

e_i Maximale Ausführungszeit (WCET)

D_i Relativer Termin (engl. deadline)

d_i Absoluter Termin

ω_i Antwortzeit (engl. response time)

σ_i Schlupf (engl. slack)

Periodische Aufgaben

p_i Periode (engl. period)

ϕ_i Phase (engl. phase)

Nicht-Periodische Aufgaben

i_j Minimale Zwischenankunftszeit
(engl. minimal interarrival-time)

Aufgaben – Tupel

$T_p = (p, e, D, \phi)$ Periodische Aufgabe ohne Priorität (zeitgesteuert oder dynamische Taskpriorität), $D = p$ und $\phi = 0$ sind der Reihe nach optional

$T_i^S = (i_j, e_j, D_j)$ Nicht-periodische Aufgabe (Schreibweise mit i_j)

$T_i^S = ([r_i^{nach}; r_i^{vor}], e_i, D_i)$ Nicht-periodische Aufgabe (Schreibweise mit Auslöseintervall)

$J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$ Arbeitsauftrag

Ablaufplanung

P_j Priorität (engl. priority) der Aufgabe T_j

Ω_j Prioritätsebenen (engl. number of priorities)

$h_{\Delta t}$ Rechenzeitbedarf (engl. demand)

$u_{\Delta t}$ CPU-Auslastung (engl. utilisation)

U Absolute CPU-Auslastung

H Hyperperiode (großer Durchlauf, engl. major cycle)

f Rahmenlänge (kleiner Durchlauf, engl. minor cycle)

e_i^f WCET aller Aufträge im Rahmen i

I_i Intervall (engl. interval)

Δ_i Dichte (engl. density) von I_i

Zusteller

T_{PS} Abfragender Zusteller (engl. polling server)

T_{DS} Aufschiebbarer Zusteller (engl. deferable server)

T_s Sporadischer Zusteller (engl. sporadic server)

T_{s} Sporadischer Zusteller (engl. sporadic server)

rt_i Wiederauffüllzeitpunkt (engl. replenishment time)

Synchronisation

cs_i Kritischer Abschnitt (engl. critical section)

R_i Betriebsmittel (engl. resource)

s_i Selbstaussetzung (engl. self suspension)

Blockierungszeit

b_i Blockierungszeit (engl. blocking time)

b_i^c — durch Wettbewerb (engl. resource contention)

b_i^{np} — durch Unverträglichkeit (engl. nonpreemptable)

b_i^{ss} — durch s_i

Prioritätsobergrenzen

\bar{P}_i Prioritätsobergrenze von R_i (engl. ceiling priority)

$\bar{P}_i(t)$ Aktuelle (System-) Prioritätsobergrenze

