

Überblick

Zugriffskontrolle

- Konkurrenz und Koordination
- Verdrängungssteuerung
- Prioritätsvererbung
- Prioritätsobergrenzen
- Zusammenfassung

Nebenläufige Aktivitäten

Nichtsequentielles Programm

Nebenläufigkeit (engl. *concurrency*) bezeichnet das Verhältnis von nicht kausal abhängigen Ereignissen, die sich also nicht beeinflussen

- ▶ Aktionen können nebenläufig ausgeführt werden, wenn keine das Resultat des anderen benötigt

```

1:   foo = 4711;
2:   bar = 42;
3:  foobar = foo + bar;
4:  barfoo = bar + foo;
5:   hal = foobar + barfoo;

```

- ▶ Zeile 1 kann nebenläufig zu Zeile 2 ausgeführt werden
- ▶ Zeile 3 kann nebenläufig zu Zeile 4 ausgeführt werden

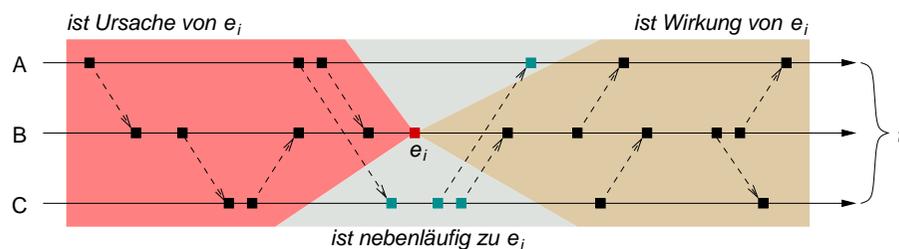
Kausalität (lat. *causa*: Ursache) ist die Beziehung zwischen **Ursache** und **Wirkung**, d.h., die ursächliche Verbindung zweier Ereignisse

- ▶ Ereignisse sind nebenläufig, wenn keines Ursache des anderen ist

Kausalordnung

Nebenläufigkeit als relativistischer Begriff von Gleichzeitigkeit

Relationen „ist Ursache von“, „ist Wirkung von“ und „ist nebenläufig zu“:



- ▶ ein Ereignis ist **nebenläufig** zu einem anderen, wenn es im **Anderswo** des anderen Ereignisses liegt
 - ▶ d.h., weder in der Zukunft noch in der Vergangenheit des anderen
- ▶ das Ereignis ist weder Ursache oder Wirkung des anderen Ereignisses

Kausalordnung (Forts.)

Rangfolge aus Gründen von Daten- und Zeitabhängigkeit

„ist Ursache von“
 „ist Wirkung von“
 „ist nebenläufig zu“ } \leadsto **Sequentialisierung** von Ereignissen/Aktionen

Aktionen können im **Echtzeitbetrieb** nebenläufig stattfinden, wenn ...

- ▶ keine das Resultat der anderen benötigt (S. 7-2) ✓
- ▶ keine die (strikten) Zeitbedingungen der anderen verletzt
 - ▶ Zeitpunkte dürfen nicht bzw. nur selten verpasst werden
 - ▶ Zeitintervalle dürfen nicht bzw. nur begrenzt zeitlich gedehnt werden
 - ▶ Abstand zwischen Ursache (Ereigniszeitpunkt) und Wirkung (Termin)

☞ die Relationen unterliegen (weichen bis harten) Echtzeitbedingungen

Koordinierung

Reihenschaltung nebenläufiger Aktivitäten

Zugriffskontrolle koordiniert nebenläufige Zugriffe auf gemeinsame aber unteilbare Betriebsmittel \leadsto Synchronisation

- ▶ typischerweise blockierend, wenn das Betriebsmittel nicht die CPU ist
- ▶ möglicherweise nicht-blockierend, sonst. . .
 - ▶ kritische Abschnitte

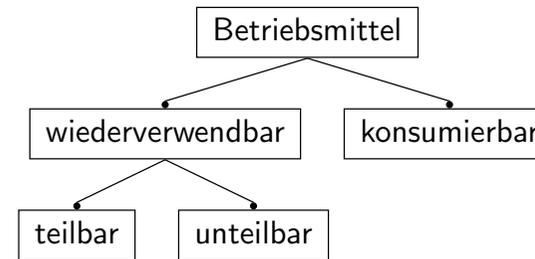
Synchronisation (gr. *syn*: zusammen, *chrónos*: Zeit) bezeichnet das „Herstellen von Gleichzeitigkeit“

- ▶ Koordination der Kooperation und Konkurrenz zwischen Prozessen
- ▶ Abgleich von Echtzeituhren (oder Daten) in verteilten Systemen
- ▶ Sequentialisierung von Ereignissen entlang einer Kausalordnung
 - ▶ z.B. logische Uhren

☞ analytisch oder konstruktiv: Einplanung vs. Implementierung (S. 4-21)

Konkurrenz

Betriebsmittel und Betriebsmittelarten



Hardware

CPU, Speicher,
Geräte, Signale

Software

Dateien, Prozesse,
Seiten, Puffer,
Signale, Nachrichten

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

einseitige Synchronisation \mapsto konsumierbare Betriebsmittel

mehrseitige Synchronisation \mapsto wiederverwendbare Betriebsmittel

- ▶ Begrenzung, gegenseitiger Ausschluss

Konkurrenz (Forts.)

Serialisierung von Arbeitsabläufen mit begrenzten/unteilbaren Betriebsmitteln

Betriebsmittel, die unteilbar sind, können von gleichzeitigen Prozessen bzw. Arbeitsaufträgen nur nacheinander belegt werden

Vergabe \mapsto das Betriebsmittel sperren und dem Job zuteilen P

- ▶ beim Versuch, ein gesperrtes Betriebsmittel erneut zu belegen, wird der anfordernde Job blockiert
- ▶ der blockierende Job erwartet das Ereignis/Signal zur Freigabe des gesperrten Betriebsmittels, ihm wird die CPU entzogen

Freigabe \mapsto das Betriebsmittel dem Job wieder entziehen V

- ▶ sollten Jobs die Freigabe dieses Betriebsmittels erwarten, wird es sofort der **Wiedervergabe** zugeführt; das bedeutet:
 - das Betriebsmittel entsperren und alle Jobs deblockieren, die sich dann wiederholen um die Vergabe zu bemühen haben *oder*
 - einen Job auswählen und ihm das Betriebsmittel zuteilen
- ▶ nur der Job, dem das Betriebsmittel zugeteilt war, kann es freigeben

Konfliktsituation

Blockierung von Arbeitsaufträgen

Arbeitsaufträge befinden sich untereinander im **Konflikt**, wenn. . .

- ▶ nur eine begrenzte Anzahl gemeinsamer Betriebsmitteln vorrätig ist
- ▶ sie unteilbare Betriebsmittel desselben Typs gemeinsam verwenden

Arbeitsaufträge sind im **Streit** (engl. *contention*) um ein Betriebsmittel, wenn einer das Betriebsmittel anfordert, das ein anderer bereits besitzt

- ▶ der anfordernde Job blockiert und wartet auf die Freigabe des Betriebsmittels durch den Job, der das Betriebsmittel belegt
- ▶ der das Betriebsmittel belegende Job löst den auf die Freigabe des Betriebsmittels wartenden Job aus, d.h., deblockiert in wieder

☞ Nutzung begrenzter/unteilbarer Betriebsmittel impliziert **Kooperation**

Wettstreit um Betriebsmittel

Beispiel: $J_l \mapsto 6(0, 18]$, $J_m \mapsto 7(2, 17]$, $J_h \mapsto 5(2, 14]$  J_l (niedrige Priorität)

t_0 startet
 t_1 belegt R_i
 t_4 setzt Ausführung fort
 t_8 setzt Ausführung fort
 t_9 gibt R_i frei $\mapsto J_h$
 t_{17} setzt Ausführung fort
 t_{18} beendet die Ausführung

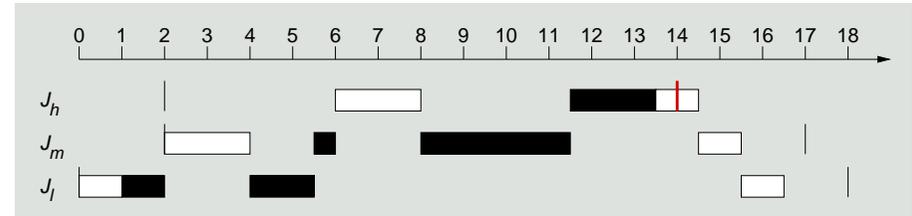
 J_m (mittlere Priorität)

t_2 startet, verdrängt J_l
 t_4 fordert R_i an, blockiert
 t_{12} belegt R_i
 t_{16} gibt R_i frei
 t_{17} beendet die Ausführung

 J_h (hohe Priorität)

t_6 startet, verdrängt J_l
 t_8 fordert R_i an, blockiert
 t_9 deblockiert, belegt R_i
 t_{11} gibt R_i frei $\leadsto J_m$
 t_{12} beendet die Ausführung

Anomalie im Laufzeitverhalten

Beispiel: $J_l \mapsto 4.5(0, 18]$, $J_m \mapsto 7(2, 17]$, $J_h \mapsto 5(2, 14]$; J_l mit kürzerer Belegungszeit J_l (niedrige Priorität)

t_0 startet
 t_1 belegt R_i
 t_4 setzt Ausführung fort
 $t_{5.5}$ gibt R_i frei $\mapsto J_m$
 $t_{15.5}$ setzt Ausführung fort
 $t_{16.5}$ beendet die Ausführung

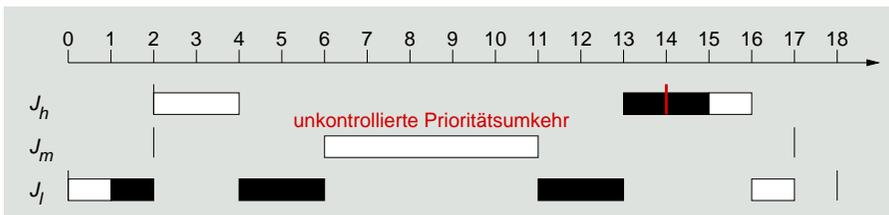
 J_m (mittlere Priorität)

t_2 startet, verdrängt J_l
 t_4 fordert R_i an, blockiert
 $t_{5.5}$ belegt R_i
 t_8 setzt Ausführung fort
 $t_{11.5}$ gibt R_i frei $\mapsto J_h$
 $t_{14.5}$ setzt Ausführung fort
 $t_{15.5}$ beendet die Ausführung

 J_h (hohe Priorität)

t_6 startet, verdrängt J_m
 t_8 fordert R_i an, blockiert
 $t_{11.5}$ belegt R_i
 $t_{13.5}$ gibt R_i frei
 t_{14} verletzt seinen Termin
 $t_{14.5}$ beendet die Ausführung

Anomalie im Laufzeitverhalten (Forts.)

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

t_0 startet
 t_1 belegt R_i
 t_4 setzt Ausführung fort
 t_{11} setzt Ausführung fort
 t_{13} gibt R_i frei $\mapsto J_h$
 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_i an, blockiert
 t_{13} belegt R_i
 t_{14} verletzt seinen Termin
 t_{15} gibt R_i frei
 t_{16} beendet die Ausführung

Synchronisation *Considered Harmful*

Prioritätsorientierte Einplanung

Prioritätsumkehr (engl. *priority inversion*, S. 6-26), mögliches Phänomen abhängiger Tasks, vorausgesetzt zwei Bedingungen sind erfüllt:

notwendig ist, dass eine Task hoher Priorität auf eine Task niedriger Priorität wartet \mapsto **gegenseitiger Ausschluss**

hinreichend ist, dass die Task niedriger Priorität von einer oder mehrerer Tasks mittlerer Priorität verdrängt wird

► auch als **unkontrollierte Prioritätsumkehr** bezeichnet

Lösungsansätze sind, sofern blockierende Synchronisation in der jeweils gegebenen Situation nicht vermieden werden kann:

- Verdrängungssteuerung ☞ S. 7-13
- Prioritätsvererbung ☞ S. 7-18
- Prioritätsobergrenzen ☞ S. 7-22

Verdrängung zeitweise unterbinden

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

Arbeitsaufträge werden für die **Gesamtzeit der Belegung** von (unteilbaren) Betriebsmitteln nicht von anderen Arbeitsaufträgen verdrängt

- ▶ die Benutzung der Betriebsmittel kontrolliert ein **Monitor** [25, 26]
 - ▶ **kernelized monitor** [27]

Eintrittsprotokoll \mapsto Verdrängung abwehren

- ▶ ausgelöste Jobs einplanen, aber nicht einlasten

Austrittsprotokoll \mapsto Verdrängung wieder zulassen

- ▶ höher priorisierte Jobs (nachträglich) einlasten

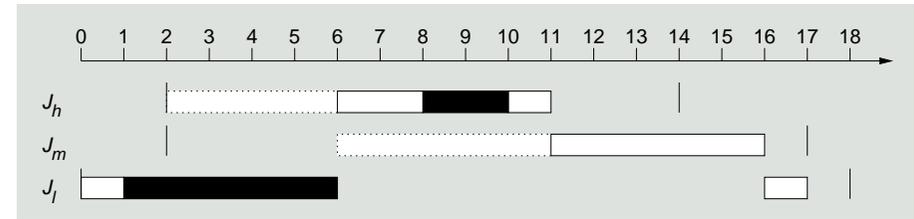
- ▶ sobald ein Job ein Betriebsmittel belegt, läuft er unverdrängbar weiter

- ▶ **verklemmungsfreies Verfahren** durch **Verklemmungsvorbeugung**
 - ▶ engl. *deadlock prevention*

☞ Taktsteuerung unterstützt die unverdrängbare Ausführung rahmenweise

Kernelized Monitor

Beispiel (S. 7-11): $J_l \mapsto 7(0, 18]$, $J_m \mapsto 5(2, 17]$, $J_h \mapsto 5(2, 14]$



J_l (niedrige Priorität)

t_0 startet

t_1 belegt R_i unverdrängbar

t_6 gibt R_i 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

J_h (hohe Priorität)

t_2 wird ausgelöst

t_6 startet

t_8 belegt R_i unverdrängbar

t_{10} gibt R_i frei

t_{11} beendet die Ausführung

Blockierungszeit (1)

Feste obere Schranke

Verzögerungen nebenläufiger Arbeitsaufträge durch die Zugriffskontrolle **sind nach oben begrenzt**

- ▶ die feste obere Schranke bt bestimmt sich aus der größten WCET aller kritischen Abschnitte aller niedriger priorisierten Jobs: $\max(cs)$
- ▶ höher priorisierte Jobs werden schlimmstenfalls einmal durch einen niedriger priorisierten Job blockiert

NPCS verzögert eine periodische Task T_i von n periodischen Tasks im taktweisen Betrieb um $bt_i = \max(cs_k)$, für $i + 1 \leq k \leq n$:

fixed-priority bei Abarbeitung nach absteigender Priorität

dynamic-priority z.B. 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 $D_i < D_j$

Pragmatischer Ansatz

Effektiv, bei vergleichsweise geringem Aufwand

Vorteil: erfordert kein *a priori* Wissen über Betriebsmittelanforderungen

- ▶ beugt unkontrollierter Prioritätsumkehr vor
 - ▶ J_h blockiert nur wenn bei Auslösung J_l bereits das Betriebsmittel hält
 - ▶ beendet J_l seinen kritischen Abschnitt, sind alle Betriebsmittel frei
 - ▶ Jobs niedrigerer Priorität als J_h können ihm diese nicht streitig machen
- ▶ beugt Verklemmung (engl. *deadlock*) vor, da Nachforderungen von Betriebsmitteln implizit unteilbar geschehen werden
 - ▶ eine notwendige Verklemmungsbedingung [3, VIII-60] wird entkräftet
 - ▶ genauer: der „*hold and wait*“ Fall kann nicht eintreten
- ▶ einfach zu implementieren; ein gutes Verfahren, wenn...
 - ▶ alle Belegungszeiten aller Betriebsmittel kurz sind
 - ▶ die meisten Jobs im Konflikt zueinander stehen

☞ eignet sich für Systeme mit fester und dynamischer Priorität

Pragmatischer Ansatz mit Schönheitsfehlern

Alternativen, sofern bestimmte Voraussetzungen gegeben sind

Nachteil: höher priorisierte Jobs können durch niedriger priorisierte Jobs blockiert werden, obwohl zwischen ihnen kein Konflikt besteht

- ▶ im Beispiel (S. 7-14) wird J_m durch J_l blockiert, obwohl beide Jobs nicht im gegenseitigen Ausschluss zueinander stehen

Verbesserungsmöglichkeiten. . .

- ▶ so Verklemmungen nicht auftreten können oder durch eine andere Technik vorgebeugt oder vermieden werden:
 - ▶ der ein Betriebsmittel haltende Job wird für die restliche Belegungszeit ggf. auf die Priorität des jeweils anfordernden Jobs hochgesetzt
 - ▶ er wird durch **Prioritätsvererbung** (S. 7-18) „beschleunigt“
- ▶ so Betriebsmittelanforderungen *à priori* bekannt sind:
 - ▶ der ein Betriebsmittel haltende Job läuft mit der höchsten Priorität aller Jobs, die das Betriebsmittel beanspruchen
 - ▶ das Betriebsmittel ist mit einer **Prioritätsobergrenze** (S. 7-22) assoziiert

Priorität zeitweise erhöhen

Wechsel zwischen zugewiesene und aktuelle (geerbte) Priorität

Arbeitsaufträge werden für die **Restzeit der Belegung** von (unteilbaren) Betriebsmitteln durch andere Arbeitsaufträge höher priorisiert

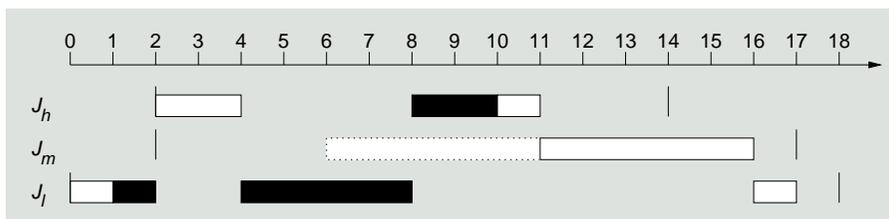
- ▶ fordert ein Job ein gesperrtes Betriebsmittel an, vererbt er seine Priorität an den das Betriebsmittel haltenden Job
 - ▶ der anfordernde Job hat zu dem Zeitpunkt die höchste Priorität
 - ▶ er hat den das Betriebsmittel haltenden Job (indirekt) verdrängt
 - ▶ die Priorität des das Betriebsmittel haltenden Jobs wird erhöht
- ▶ gibt der Job das durch ihn gesperrte Betriebsmittel frei, nimmt er die ihm ursprünglich zugewiesene Priorität wieder an
 - ▶ der das Betriebsmittel freigebende Job wird ggf. sofort verdrängt
 - ▶ der auf die Freigabe wartende Job wird ggf. sofort eingelastet

Prioritätsumkehr wird nicht wirklich vermieden, jedoch entschärft:

- ▶ wie bei NPCS behalten niedriger priorisierte Jobs die CPU zugeteilt, obwohl höher priorisierte Jobs auf Zuteilung der CPU warten

Priority Inheritance

Beispiel (S. 7-11): $J_l \mapsto 7(0, 18]$, $J_m \mapsto 5(2, 17]$, $J_h \mapsto 5(2, 14]$



J_l (niedrige Priorität)

t_0 startet

t_1 belegt R_i

t_4 läuft mit Priorität von J_h

t_8 gibt R_i frei $\mapsto J_h$

t_{16} läuft mit alter Priorität

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 startet

t_4 fordert R_i an $\mapsto J_l$

t_8 belegt R_i

t_{10} gibt R_i frei

t_{11} beendet die Ausführung

Transitive Blockierung

Nachforderung unteilbarer Betriebsmittel

Zugriffskontrolle durch Prioritätsvererbung bedeutet zunächst zweierlei:

direkte Blockierung eines höher priorisierten Jobs (J_h), der ein gesperrtes Betriebsmittel anfordert, das ein niedriger priorisierter Job (J_l) belegt

Blockierung durch Vererbung (engl. *inheritance blocking*) eines nicht im gegenseitigen Ausschluss befindlichen höher priorisierten Jobs (J_m)

- ▶ der einen Job (J_l) gemäß dessen „Altpriorität“ verdrängen würde
- ▶ dies jedoch wegen dessen aktuellen (geerbten) Priorität nicht kann

Blockierungen wirken ggf. transitiv: **geschachtelte kritische Abschnitte**

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

Blockierungszeit (2)

Feste obere Schranken, die kaskadenartig zur Wirkung kommen können

Arbeitsaufträge höherer Priorität werden nur einmal direkt blockiert, wenn die Nachforderung unteilbarer Betriebsmittel ausgeschlossen ist

- ▶ für die Dauer der WCET des äußersten kritischen Abschnitts

Situation des schlimmsten Falls: ein höher priorisierter Job benötigt $n > 1$ Betriebsmittel und steht mit $k > 1$ niedriger priorisierten Jobs im Konflikt

- ▶ der höher priorisierte Job kann $\min(n, k)$ -mal blockiert werden
- ▶ jeweils für die Dauer der WCET des äußersten kritischen Abschnitts

GAU: $n > 1$ Betriebsmittel; $k > 1$ Jobs; J_i hat Vorrang vor J_j , wenn $i < j$

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 $i = 1, 2, 3, \dots, n$

Priorität zeitweise deckeln

Vorwissen über Arbeitsaufträge und Betriebsmittel

Prioritätsobergrenze (engl. *priority ceiling*) eines Betriebsmittels R_i ist die höchste Priorität aller Arbeitsaufträge, die R_i benötigen

- ▶ die **aktuelle Prioritätsobergrenze** des Systems gleicht der höchsten Prioritätsobergrenze der zur Zeit belegten Betriebsmittel
 - ▶ $\hat{\Pi}(t)$, in Abhängigkeit vom betrachteten Zeitpunkt t
- ▶ ist kein unteilbares Betriebsmittel belegt, existiert die aktuelle Prioritätsobergrenze (theoretisch) nicht
 - ▶ sie ist dann niedriger als die niedrigste Priorität aller Jobs
- ▶ die jeweiligen Werte sind für alle Betriebsmittel im Voraus bekannt

Arbeitsaufträge können während ihrer Ausführung (d.h., bei Anforderung eines gesperrten Betriebsmittels) eine durch x begrenzte Priorität erben

- ▶ wenn sie ein Betriebsmittel mit Prioritätsobergrenze x benötigen

☞ Prioritätsobergrenzen sind eine Variante von Prioritätsvererbung

Betriebsmittelvergabe und Prioritätsvererbung

Vorwissen über Arbeitsaufträge und Betriebsmittel

Vergabe von Betriebsmittel R zum Zeitpunkt t an Arbeitsauftrag J hängt ab vom Zustand von R und von der aktuellen Priorität $\pi(t)$ von J :

belegt $\mapsto R$ ist gesperrt, J blockiert

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

1. $\pi(t) > \hat{\Pi}(t)$: J hat die aktuell höchste Priorität
2. $\pi(t) \leq \hat{\Pi}(t)$: J ist ein Job, der zum Zeitpunkt t mindestens ein Betriebsmittel mit Prioritätsobergrenze $\hat{\Pi}(t)$ hält
 - ▶ anderenfalls bleibt R frei und J blockiert (S. 7-24)

Prioritätsvererbung findet (auch hier) nur statt, wenn J suspendiert wird:

- ▶ J_i , der J blockiert, erbt die aktuelle Priorität $\pi(t)$ von J
- ▶ J_j behält diese Priorität, bis er alle Betriebsmittel freigibt, deren Prioritätsobergrenze größer oder gleich $\pi(t)$ ist
 - ▶ er nimmt dann Priorität $\pi(t')$ an, die er zum Zeitpunkt $t' < t$ hatte
 - ▶ d.h., als ihm die freigegebenen Betriebsmittel zugeteilt wurden

Verklemmungsvorbeugung

Entkräftung der hinreichenden Bedingung [3, VIII-60]: zirkulares Warten

Betriebsmittelvergabe kontrolliert durch Prioritätsobergrenzen ist weniger „gefräßig“ (engl. *greedy*), als bloße Prioritätsvererbung

- ▶ die Anforderung von J kann zurückgewiesen werden, obwohl das angeforderte Betriebsmittel R frei ist
- ▶ dies ist der Fall, wenn die durch die Menge von Prioritätsobergrenzen definierte (ansteigende) **lineare Ordnung** verletzt werden sollte
 - ▶ $\pi(t) \leq \hat{\Pi}(t)$ trifft zu und J hält kein Betriebsmittel mit $\hat{\Pi}(t)$
 - ▶ d.h., 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

- ▶ da implizit die Kosten anfallen, um auch Verklemmungen vorzubeugen

Blockierungszeit (3)

Feste obere Schranke

Zugriffskontrolle durch Prioritätsobergrenzen impliziert drei Arten der Blockierung nebenläufiger Arbeitsaufträge:

1. direkte Blockierung,
 2. Blockierung durch Vererbung,
 3. Blockierung in Form einer Aufhebungssperre
- } **Prioritätsvererbung**

Effekt von 3. ist, dass jeder Arbeitsauftrag höchstens einmal blockiert und dass eine Blockierung nicht transitiv ist [28]

- ▶ die Blockierungszeit ist begrenzt durch die größte WCET aller kritischen Abschnitte aller niedriger priorisierten Jobs
- ▶ unabhängig von der Anzahl der im Konflikt stehenden Jobs
 - (a) wenn ein Job blockiert, dann nur durch höchstens einen Job
 - (b) ein Job, der einen anderen blockiert, wird von keinem anderen blockiert

Bestimmung von Blockierungszeiten

Betriebsmittelanforderungsgraph (engl. *resource requirement graph*)

Ansatz zur Bestimmung von Blockierungszeiten ist ein **Wartegraph**, der Jobs mit Betriebsmitteln und umgekehrt in Verbindung bringt¹⁸

Knoten repräsentieren Jobs bzw. Betriebsmittel, annotiert mit einem Namen und der Betriebsmittelanzahl (nur Betriebsmittelknoten)

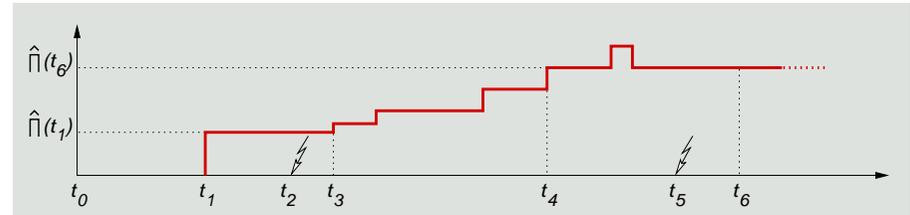
Kanten drücken „wartet auf“- bzw. „belegt von“-Beziehungen aus und sind mit einem 2-Tupel (n, t) annotiert

- ▶ Betriebsmittelanzahl n , Belegungszeit t
- ▶ Pfade zwischen Jobknoten führen immer über Betriebsmittelknoten
 - ▶ J_x belegt n Einheiten R_i für t Zeiten, worauf J_y wartet
 - ▶ führt ein Pfad von J_h zu J_l , ist J_h durch J_l direkt blockiert
- ▶ Zyklen im Wartegraph zeigen **Verklemmungen** an \rightsquigarrow „Abfallprodukt“ **Verklemmungserkennung** (engl. *deadlock detection*) beim Entwurf
 - ▶ Verklemmungsvorbeugung zur Laufzeit als Zusicherung (engl. *assertion*)

¹⁸Wenn Verklemmungserkennung gleich mit erledigt werden soll. Ansonsten reicht es, den längsten kritischen Abschnitt aller niedrigeren Tasks zu finden.

Priority Ceiling

Jobs blockieren nicht transitiv und höchstens einmal im Konfliktfall



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 $R_x \rightsquigarrow \hat{\pi}(t_1)$	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 π_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 ein Job 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

Vereinfachung durch Stapelorientierung

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

Ausgangspunkt ist die **stapelorientierte Einplanung von Prozessen**, so dass N Prozesse einen Stapel gemeinsam nutzen können [29, 30]

- ▶ nicht immer ist es möglich, jeden Job durch einen eigenen Faden zu repräsentieren bzw. mit einem eigenen Stapel zu versehen
 - ▶ wenn die Jobanzahl zu hoch und/oder der Speicherplatz zu gering ist
- ▶ gemeinsame Nutzung desselben Stapels setzt voraus, dass kein Job bei Anforderung eines gemeinsamen Betriebsmittels blockiert
 - ▶ sonst droht die Überschreitung der Stapelbereiche anderer Jobs
- ▶ die **Jobs dürfen ihre Ausführung niemals selbst aussetzen**, sie dürfen jedoch von höher priorisierten Jobs verdrängt werden
 - ▶ oben auf dem Stapel läuft immer der Job mit der höchsten Priorität
 - ▶ die logische Konsequenz, wenn Selbstausssetzung ausgeschlossen ist

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

Stapelbezogene Grenzprioritäten

Regeln

1. Aktualisierung der **Grenzpriorität** (engl. *ceiling priority*, S. 7-22) $\hat{\pi}(t)$
 - ▶ erfolgt mit jeder Vergabe/Freigabe von Betriebsmitteln
 - ▶ sind alle Betriebsmittel frei, gibt es keine Grenzpriorität
2. Einplanung und Einlastung von Jobs
 - ▶ nach erfolgter Auslösung muss ein Job ggf. solange warten, bis die ihm zugewiesene Priorität die Grenzpriorität übersteigt
 - ▶ Jobs werden jeweils entsprechend ihrer zugewiesenen Priorität und verdrängend ausgeführt
3. Zuteilung eines Betriebsmittels
 - ▶ erfolgt sofort, wann immer ein Job das Betriebsmittel anfordert

Jobs blockieren nicht, indem ihnen die Betriebsmittelzuteilung nach Ausführungsbeginn verweigert werden würde

- ▶ im Gegensatz zum Grundmodell von Prioritätsobergrenzen (S. 7-23)

Stapelbezogene Grenzprioritäten (Forts.)

Implikationen

Verklebungen nebenläufig ausgeführter Arbeitsaufträge sind durch eine **indirekte Methode zur Verklebungsvorbeugung**¹⁹ ausgeschlossen

- (a) beginnt ein Job seine Ausführung, sind alle von ihm im weiteren Verlauf benötigten Betriebsmittel frei
 - ▶ sonst wäre die Grenzpriorität größer oder gleich seiner Priorität
 - ▶ in dem Fall wäre jedoch die Einlastung des Jobs verzögert worden
- (b) bei Verdrängung eines Jobs 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

¹⁹ **zirkulares Warten** wird vorgebeugt (siehe auch S. 7-24)

Grundmodell und Stapelorientierung

Feste vs. dynamische Priorität (vergleiche S. 6-6)

feste Priorität \mapsto einfach, wegen *à priori* Wissen

dynamische Priorität \mapsto Prioritäten periodischer Aufgaben ändern sich, während die von ihnen beanspruchten Betriebsmittel konstant bleiben:

☞ **Grenzprioritäten der Betriebsmittel ändern sich mit der Zeit**

- ▶ Aktualisierung der Grenzprioritäten bei jeder Auslösung eines Jobs
 1. dem ausgelösten Job zum Ereigniszeitpunkt eine Priorität zuweisen
 - ▶ relativ zu den anderen bereits eingeplanten/laufbereiten Jobs
 - ▶ (dynamische) prioritätsorientierte Einplanung je nach Verfahren
 2. Grenzprioritäten aller Betriebsmittel aktualisieren
 - ▶ auf Basis der neuen Taskprioritäten
 3. Grenzpriorität des Systems Betriebsmittel aktualisieren
 - ▶ auf Basis der neuen Grenzprioritäten der Betriebsmittel
- ▶ bei auf Jobebene statische wie auch dynamische Prioritäten (S. 6-7)

Resümee

Konkurrenz und Koordination nebenläufiger Aktivitäten

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

Verdrängungssteuerung \mapsto verdrängungsfreie kritische Abschnitte

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

Prioritätsvererbung \mapsto Priorität zeitweise erhöhen

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

Prioritätsobergrenzen \mapsto Priorität zeitweise deckeln

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