

Phase 4: Komposition – ProOSEK

Echtzeitsysteme 2 - Vorlesung/Übung

Fabian Scheler
Michael Stilkerich
Peter Ulbrich
Wolfgang Schröder-Preikschat

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

<http://www4.cs.fau.de/~{scheler,mike,wosch}>
{scheler,mike,ulbrich,wosch}@cs.fau.de



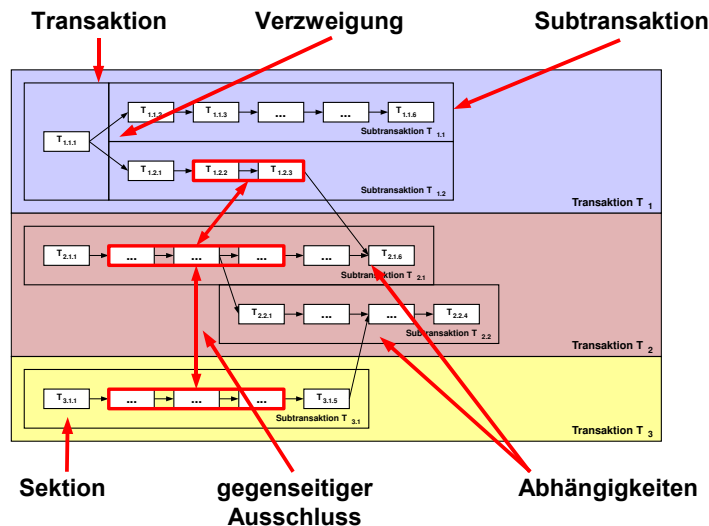
1

Übersicht

- Wiederholung
 - Ereignisbehandlungen
 - Abbildung
 - Antwortzeitanalyse
- ProOSEK
 - Abbildung
 - Antwortzeitanalyse
 - Hinweise

2

Wiederholung: Ereignisbehandlungen



3

Wiederholung: Abbildung

- Implementierung von
 - Transaktionen, Subtransaktionen und Sektionen
 - Konkatenation von Sektionen
 - gegenseitigem Ausschluss
 - Verzweigungen
 - Abhängigkeiten
- Implementierung mit den Mitteln des Betriebssystems

4

Wiederholung: Antwortzeitanalyse

- **Auslösezeitpunkt:** Zeitpunkt an dem ein Ereignis eintritt
- **Antwortzeit:** Zeitpunkt an dem die Behandlung des Ereignisses abgeschlossen ist
 - mit anderen Worten: eine Subtransaktion

maximale Antwortzeit:

$$w_i(t) = e_i + \sum_{k=1}^{i-1} \text{ceiling}\left(\frac{t}{p_k}\right) e_k$$

$$w_i(t) \leq t; t = jp_k; k = 1, 2, \dots, i; j = 1, 2, \dots, \text{floor}(\min(d_i, p_i) / p_k)$$

- hinreichendes und notwendiges **Kriterium für Planbarkeit:**

$$\max_i w_i(t) \leq d_i \forall \text{Task } t_i$$

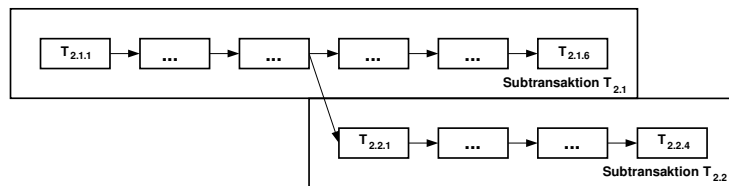


ProOSEK: Abbildung

- Verzweigungen
- Abhängigkeiten
- aperiodische Ereignisbehandlungen



ProOSEK: Verzweigungen



- Transaktion spaltet sich in unabhängige Kontrollflüsse
- Subtransaktionen münden nicht wieder zurück

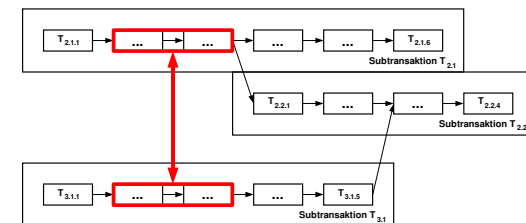
```

TASK(A) {
    ...
    ActivateTask(B);
    ...
    ChainTask(C);
}
    
```

- eine Subtransaktion entsteht nur wenn
 - Task A nicht-präemptiv ist oder
 - Priorität(B) < Priorität(A)
- nieder-priore Subtransaktionen zweigen immer von hoch-prioren Subtransaktionen ab



ProOSEK: gegenseitiger Ausschluss



- Verdrängungssteuerung (nicht verdrängbarer Task)
 - TASK B: nicht-präemptiv und Prio(B) > Prio(A)
 - TASK A: verdrängbar
 - × Overhead durch Kontextwechsel

```

TASK(A) {
    ...
    ActivateTask(B);
    ...
    TerminateTask();
}

TASK(B) {
    ...
    criticalStuff();
    ...
    TerminateTask();
}
    
```



ProOSEK: gegenseitiger Ausschluss

■ Verdrängungssteuerung (*Kernelized Monitor*)

- spezielle Resource: RES_SCHEDULER
- verhindert Verdrängung komplett
- Synchronisation nur auf TASK-Ebene
- ✓ kein Kontextwechsel
- ✗ Blockade potentiell unbeteiligter TASKs

```
TASK(A) {  
    ...  
    GetResource(RES_SCHEDULER);  
    doCriticalStuff();  
    ReleaseResource(RES_SCHEDULER);  
    ...  
    TerminateTask();  
}
```



ProOSEK: gegenseitiger Ausschluss

■ Schlossvariablen (RESOURCEn)

- OSEK Priority Ceiling Protocol (= *Stack-based Priority Ceiling*)
- RESOURCEn statisch konfiguriert
- Implementierung (optional) auch für ISR Cat. 2
- ✓ keine unnötigen Blockaden
- ✗ für ISR Cat. 2 häufig mittels Suspend-/ResumeOSInterrupts()

```
TASK(A) {  
    ...  
    GetResource(LOCK);  
    doCriticalStuff();  
    ReleaseResource(LOCK);  
    ...  
    TerminateTask();  
}  
  
ISR(ISRA) {  
    ...  
    GetResource(LOCK);  
    doCriticalStuff();  
    ReleaseResource(LOCK);  
    ...  
    return;  
}
```



ProOSEK: gegenseitiger Ausschluss

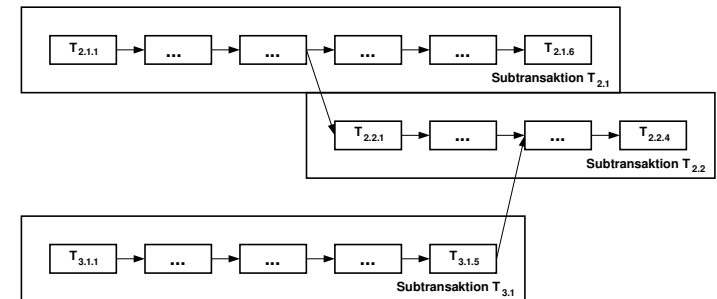
■ harte Synchronisation

- sperren der Unterbrechungen
- Selektiv:
 - 1)ISR Cat. 1 + ISR Cat. 2 → Enable-/DisableAllInterrupts()
 - 2)ISR Cat. 2 → Resume-/SuspendOSInterrupts()
- immer **paarweise**, Schachtelung möglich
- ✗ verhindert jegliche Entgegennahme von Ereignissen
 - Ereignisse können verloren gehen (auch Timerinterrupts)

```
TASK(A) {  
    ...  
    SuspendOSInterrupts();  
    doCriticalStuff();  
    ResumeOSInterrupts();  
    ...  
    TerminateTask();  
}  
  
ISR(ISRA) {  
    ...  
    DisableAllInterrupts();  
    doCriticalStuff();  
    EnableAllInterrupts();  
    ...  
    return;  
}
```



ProOSEK: Abhängigkeiten



■ Arten von Abhängigkeiten

- ODER-Abhängigkeiten
- UND-Abhängigkeiten
- Nachfolger hängt von mehreren Vorgängern ab



ProOSEK: ODER-Abhängigkeiten

- **Ansatz:** mehrfache Auslösbarkeit des Nachfolgers
 - jeder Vorgänger löst jeweils den Nachfolger aus
- **zu beachten:**
 - Prioritätenvergabe
 - Anzahl der Auslösungen



ProOSEK: UND-Abhängigkeiten

- **Ansatz:** EVENTS
 - Nachfolger wartet zyklisch bis mehrere EVENTS gesetzt sind
 - Vorgänger setzen jeweils ein EVENT

```
Task(Successor) {
    EventMaskType eventsAwaited;
    EventMaskType eventsNotSet;
    eventsAwaited = e1 | e2 | e3;
    eventsNotSet = eventsAwaited;

    do {
        EventMaskType eventsSet = 0;
        WaitEvent(eventsNotSet);
        GetEvent(Successor, &eventsSet);
        eventsNotSet ^=
            (eventsSet & eventsAwaited);
    } while(eventsNotSet != 0);
    ...
    TerminateTask();
}

Task(Predecessor1) {
    ...
    SetEvent(Successor, e1);
    TerminateTask();
}

Task(Predecessor2) {
    ...
    SetEvent(Successor, e2);
    TerminateTask();
}

Task(Predecessor3) {
    ...
    SetEvent(Successor, e3);
    TerminateTask();
}
```



ProOSEK: UND-Abhängigkeiten

- **zu beachten:**
 - Overhead durch Kontextwechsel
 - erschwert Antwortzeitanalyse
 - Prioritätenvergabe



ProOSEK: UND-Abhängigkeiten

- **Ansatz:** ALARM & USERCOUNTER
 - Nachfolger wird von einem ALARM ausgelöst
 - ALARM wird durch einen USERCOUNTER gesteuert
 - Vorgänger inkrementieren den USERCOUNTER

```
TASK(Successor) {
    ...
    SetRelAlarm(alarm, 3, 0);
    TerminateTask();
}

Task(Predecessor2) {
    ...
    AdvanceCounter(counter);
    TerminateTask();
}

Task(Predecessor1) {
    ...
    AdvanceCounter(counter);
    TerminateTask();
}

Task(Predecessor3) {
    ...
    AdvanceCounter(counter);
    TerminateTask();
}
```



ProOSEK: UND-Abhängigkeiten

- **zu beachten:**
 - Prioritätenvergabe
 - Wann soll der ALARM ablaufen?
 - Vorgänger können unterschiedliche Periode besitzen
 - der ALARM muss initialisiert werden



ProOSEK: aperiodische Ereignisse

- verfügbare Methoden: *deferred handling*
- Vorgehen
 - Ereignis wird initial durch ISR behandelt
 - ISR löst eine TASK aus
 - Auslösung *speichert* das Ereignis
- nicht implementierbar:
 - *deferred server*
 - *sporadic server*
 - ...
- zu beachten:
 - minimale Zwischenankunftszeit
 - pessimistische Antwortzeitanalyse
 - Anzahl der Auslösungen



ProOSEK: Ablaufplanung

- Begriffe
- Algorithmus
- einfache Antwortzeitanalyse
- erweiterte Antwortzeitanalyse



Begriffe

- Transaktionen, Subtransaktionen, Sektionen
- **Ausführungspfad**

Ein Ausführungspfad ist ein Kontrollfluss innerhalb einer Subtransaktion und besteht aus Sektionen die sequentiell hintereinander ausgeführt werden. Dieselbe Sektion kann auch mehrmals in einem Ausführungspfad auftreten.
- **Abschnitt**

Ein Abschnitt ist eine Menge von Sektionen, die in einem Ausführungspfad direkt aufeinander folgen.
- WCET der Subtransaktion $T_{n,m}$:

$$e_{n,m} = \max_{Pa_{n,m,l} \in T_{n,m}} \sum_{T_{n,m,i} \in Pa_{n,m,l}} e_{n,m,i} k_{n,m,i}(T_{n,m,i})$$

$Pa_{n,m,l}$ Pfad l der Subtransaktion $T_{n,m}$
 $k_{n,m,l}(T_{n,m,i})$ Auftreten der Sektion $T_{n,m,i}$ im Pfad $Pa_{n,m,l}$



Begriffe

- Deadline, Periode, minimale Zwischenankunftszeit
- Planbarkeit
- **kritischer Zeitpunkt**

Eine Subtransaktion erreicht ihre **maximale Antwortzeit** genau dann, wenn sie ihre maximale Ausführungszeit erreicht und an einem **kritischen Zeitpunkt** ausgelöst wird.



Algorithmus

- OSEK
 - Vorrangsteuerung
 - MLQ-Scheduler
 - statisch bestimmte Prioritäten
- zur Auswahl stehende Algorithmen
 - RMA
 - DMA
- Entscheidung: DMA
 - Begründung: weniger restriktiv als RMA



Einfach Ablaufplanung

- trivialer Fall
- TASKs
- kritische Abschnitte
- ALARMe
- Self-Suspension
- beliebige Deadlines



Trivialer Fall

- Ereignisse
 - strikt periodisch
 - Deadline kleiner oder gleich Periode
 - keine Abhängigkeiten
- OSEK
 - ausschließlich ISRs Cat. 1 und Cat. 2
 - ISRs verhalten sich präemptiv
 - kein ISR sperrt Interrupts
 - ausreichende Anzahl von Prioritätsebenen
 - keine
 - TASKs
 - RESOURCEn
 - ALARMe
 - EVENTs



Trivialer Fall

■ allgemein:

$$r_{n,m} = e_{n,m} + i_{n,m} + b_{n,m}$$

$i_{n,m}$ Zeit in der $T_{n,m}$ durch höher-priore Subtransaktionen unterbrochen wird

$b_{n,m}$ Zeit in der $T_{n,m}$ durch nieder-priore Subtransaktionen blockiert wird

■ hier:

$$i_{n,m} = \sum_{T_{ij} \in H_{n,m}} \text{ceiling}(r_{n,m}/p_i) e_{ij}$$

$H_{n,m}$ Menge der höher-prioren Subtransaktionen

$\text{ceiling}(r_{n,m}/p_i)$ Auftreten des Ereignisses T_i während der Ausführung der Subtransaktion $T_{n,m}$

■ insgesamt:

$$r_{n,m} = e_{n,m} + \sum_{T_{ij} \in H_{n,m}} \text{ceiling}(r_{n,m}/p_i) e_{ij} + b_{n,m}$$



TASKs

- keine nicht-präemptiven TASKs
- kein TASK sperrt die Interrupts
- ein TASK aktiviert maximal einen Nachfolger
- keine RESOURCEn, EVENTs oder Schedule()

■ Problematik:

- **ISRs nieder-priorer Ereignisse** unterbrechen
- **TASKs hoch-priorer Ereignisse**



TASKs

■ hier:

$$i_{n,m} = \underbrace{\sum_{T_{ij} \in H_{n,m}} \text{ceiling}(r_{n,m}/p_i) e_{ij}}_{(1)} + \underbrace{\sum_{T_{ij} \in L_{n,m}} \text{ceiling}(r_{n,m}/p_i) e_{ij}^{ISR}}_{(2)}$$

(1) Verdrängung durch höher-priore Subtransaktionen

(2) Verdrängung durch ISRs nieder-priorer Subtransaktionen

$L_{n,m}$ Menge der nieder-prioren Subtransaktionen

e_{ij}^{ISR} Ausführungszeit des ISR-Anteils der Subtransaktion $T_{n,m}$



Kritische Abschnitte

■ Implementierungsvarianten

- gesperrte Interrupts
- nicht-präemptive TASKs
- RESOURCEn

■ hier:

$$b_{n,m} = b_{n,m}^{non} + b_{n,m}^{ISR} + b_{n,m}^{res}$$

$b_{n,m}^{non}$ Blockade durch nicht-präemptive TASKs

$b_{n,m}^{ISR}$ Blockade durch gesperrte Interrupts

$b_{n,m}^{res}$ Blockade durch belegte RESOURCEn



Kritische Abschnitte

- Begründung
 - TASK A belegt eine RESOURCE
 - TASK A wird von einem nicht-präemptiven TASK B verdrängt
 - ISR C unterbricht TASK B und sperrt die Interrupts
- Bestimmung der einzelnen Komponenten durch
 - Sektionen
 - Ausführungspfade
 - Abschnitte



ALARMe

- ALARM werden von COUNTERn gesteuert
 - USERCOUNTER betrachten wir hier nun nicht
 - hier: TIMERCOUNTER
- TIMERCOUNTER
 - Hardware-Timer
 - löst zyklisch Interrupt aus
 - je nach Zählerstand läuft ALARM ab
- Problem
 - Timer-ISR
 - unterbricht andere Subtransaktionen
 - welche Priorität soll die ISR haben



ALARMe

- insgesamt: $w_{n.m}(t) = e_{n.m} + t c_{n.m}^{isr}(t) + i_{n.m} + b_{n.m}$
 $t c_{n.m}^{isr}(t)$ Unterbrechungen der Subtransaktion $T_{n.m}$ durch Timer-ISRs
- hier: $t c_{n.m}^{isr}(t) = \sum_{t \in TC} t c_{n.m}^{tc}(t)$
 $t c_{n.m}^{tc}(t) = 0$ Subtransaktion $T_{n.m}$ enthält keine TASKs und alle ISRs haben eine höhere Priorität als der ISR des Hardware-Timers
 $t c_{n.m}^{tc}(t) = \text{ceiling}(t / p_{TC}) e_{TC}^{isr}$ sonst
 TC Menge aller Timer-Counter
- Priorität der Timer-ISR
 - so hoch wie nötig, so niedrig wie möglich
 - es darf **kein** Tick verloren gehen
 - iterative Annäherung



Self-Suspension

- in OSEK durch
 - Warten auf ein EVENT: `WaitEvent()`
 - Aufruf des Schedulers: `Schedule()`
- Problematik:
 - Blockade durch nieder-priore Subtransaktionen kann nach jedem Verzicht auf den Prozessor erneut auftreten
- insgesamt: $w_{n.m}(t) = e_{n.m} + t c_{n.m}^{isr}(t) + i_{n.m} + s_{n.m} + (s_{n.m}^k + 1) b_{n.m}$
 $s_{n.m}$ so lange verzichtet Subtransaktion $T_{n.m}$ auf den Prozessor
 $s_{n.m}^k$ so oft verzichtet Subtransaktion $T_{n.m}$ auf den Prozessor



Self-Suspension

- Alternative: aktives Warten

```
Task(UglyTask) {  
    ...  
    SetRelAlarm(alarm1, time_to_wait, 0);  
    while(GetAlarm(alarm1) != E_OS_NO_FUNC);  
    ...  
    TerminateTask();  
}
```

- Problematisch

- erhöht Prozessorauslastung unnötigerweise
- nieder-priore Subtransaktionen werden lange verzögert
- Kann zu **Lifelocks** führen

→ Solche Probleme auf Entwurfsebene vermeiden



beliebige Deadlines

- bisher: $\text{Deadline} \leq \text{Periode}$

- jetzt: $\text{Deadline} > \text{Periode}$

- Ereignis kann erneut eintreten bevor es fertig behandelt wurde
- mehrere Inkarnationen einer Subtransaktionen können existieren
- TASKs **müssen mehrfach aktivierbar** sein
- ISRs **können nicht mehrfach** aktiviert werden

- Analyse am kritischen Zeitpunkt reicht nicht mehr aus

- Analyse muss sich über ein **Auslastungsintervall** erstrecken

Ein **Priorität-A-Auslastungsintervall** $[a,b]$ ist ein Zeitintervall in dem nur Subtransaktionen mit einer Priorität $\geq A$ ausgeführt werden. In den Zeiträumen $]a - \epsilon, a[$ und $]b, b + \epsilon[$ werden nur Subtransaktionen mit einer Priorität $< A$ ausgeführt.



Erweiterte Ablaufplanung

- Laufzeitprioritäten
- Verzweigungen
- abhängige Ereignisse



Laufzeitprioritäten

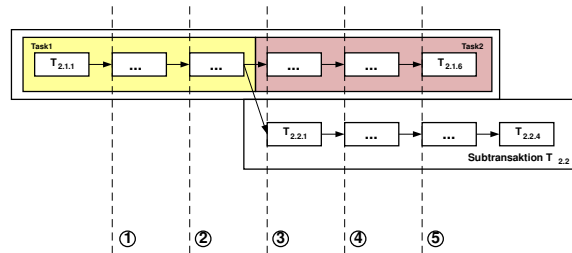
- bisherige Verwendung von Sektionen

- Berechnung der WCET
- Berechnung der Belegungszeit von RESOURCEn
- Sperrung von Unterbrechungen

- jetzt: Priorität kann zwischen Sektionen variieren

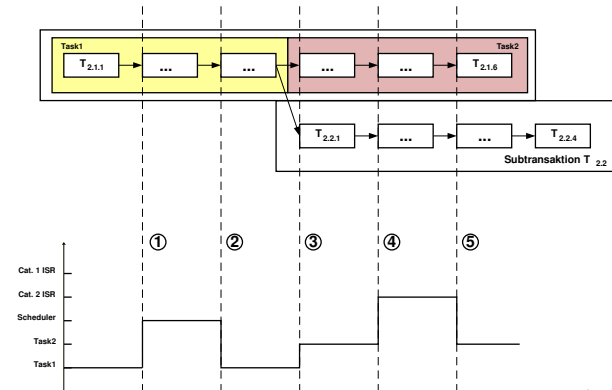


Laufzeitprioritäten



1	GetResource(RES_SCHEDULER)
2	ReleaseResource(RES_SCHEDULER)
3	ChainTask(Task2)
4	Suspend0SInterrupts()
5	Resume0SInterrupts()

Laufzeitprioritäten



1	GetResource(RES_SCHEDULER)
2	ReleaseResource(RES_SCHEDULER)
3	ChainTask(Task2)
4	Suspend0SInterrupts()
5	Resume0SInterrupts()

Laufzeitprioritäten: Antwortzeitanalyse

Begriffe

Unterbrechungsblock einer Sektion

Ein **Unterbrechungsblock** einer Sektion ist ein längst möglicher **Abschnitt eines Ausführungspfades** dessen Sektionen alle eine **größere Priorität** haben als diese Sektion.

führender Unterbrechungsblock

Ein **führender Unterbrechungsblock** ist ein Unterbrechungsblock dessen erste Sektion zugleich die **erste Sektion eines Ausführungspfades** ist.

Verzögerungsblock einer Sektion

Ein **Verzögerungsblock** einer Sektion ist ein längst möglicher **Abschnitt eines Ausführungspfades** dessen Sektionen alle eine **größere/gleich Priorität** haben als/wie diese Sektion.

Laufzeitprioritäten: Antwortzeitanalyse

Klassen von Subtransaktionen:

$L_{n,m,l}$

Alle Subtransaktionen enthalten keinen Unterbrechungsblock der Sektion $T_{n,m,l}$.

$U_{n,m,l}$

Alle Subtransaktionen enthalten einen führenden Unterbrechungsblock der Sektion $T_{n,m,l}$.

$V/L_{n,m,l}$

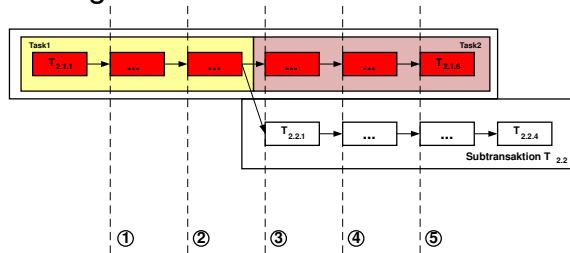
Alle Subtransaktionen enthalten einen mindestens einen Verzögerungsblock der Sektion $T_{n,m,l}$.

Zweck

- führender Unterbrechungsblock: **mehrmalige Unterbrechung**
- Unterbrechungs-/Verzögerungsblock: **einmalige Unterbrechung**

Laufzeitprioritäten: Antwortzeitanalyse

- Analyse erfolgt für einen bestimmten Ausführungspfad



- alle Ausführungspfade müssen analysiert werden
- Analyse erfolgt sukzessive
 - Berechnung der Antwortzeit der ersten Sektion
 - Berechnung der Antwortzeit der zweiten Sektion ausgehend von diesem Ergebnis



Laufzeitprioritäten: Antwortzeitanalyse

- Antwortzeit der Sektion $T_{n.m.1}$

$$w_{n.m.1}^k(t^{(l)}) = b_{n.m.1} + e_{n.m.1} + (k-1)e_{n.m.1} + \sum_{T_{ij} \in U_{n.m.1}} \text{ceiling}\left(\frac{t^{(l)}}{p_i}\right) e_{ij,n.m.1}^u + \sum_{T_{ij} \in V/L_{n.m.1}} e_{ij,n.m.1}^u$$



Laufzeitprioritäten: Antwortzeitanalyse

- Antwortzeit der Sektion $T_{n.m.1}$

Blockade

wiederholte Verzögerung durch führende Unterbrechungsblöcke

$$w_{n.m.1}^k(t^{(l)}) = b_{n.m.1} + e_{n.m.1} + (k-1)e_{n.m.1} + \sum_{T_{ij} \in U_{n.m.1}} \text{ceiling}\left(\frac{t^{(l)}}{p_i}\right) e_{ij,n.m.1}^u + \sum_{T_{ij} \in V/L_{n.m.1}} e_{ij,n.m.1}^u$$

WCET der Unterbrechungs – bzw. Verzögerungsblöcke

Zeitbedarf von $T_{n.m.1}$ und der (k-1) vorherigen Auftreten von T_n

einmalige Verzögerungen durch Verzögerungsblöcke



Laufzeitprioritäten: Antwortzeitanalyse

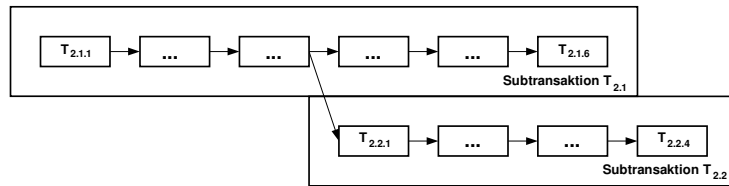
- Antwortzeit der Sektion $T_{n.m.r}$, $r > 1$

$$w_{n.m.r}^k(t^{(l)}) = \underbrace{f_{n.m.(r-1)}^k + e_{n.m.r}}_{(1)} + (2) + (3) + (4) + (5)$$

- (1) Antwortzeit der vorhergehenden Sektion und WCET der gegenwärtigen Subtransaktion
- (2) Unterbrechung durch führende Unterbrechungsblöcke – unter bestimmten Umständen ($\text{Prio}(T_{n.m.1}) < \text{Prio}(T_{n.m.(r-1)})$) können sich Aktivierungen für Unterbrechungsblöcke aufstauen – diese aufgestauten Aktivierungen werden hier nicht berücksichtigt
- (3) Aufgestaute, frühere Aktivierungen führender Unterbrechungsblöcke
- (4) Verzögerungsblöcke, die seit Beginn der Subtransaktion aktiv sind und nicht durch einen vorhergehenden Unterbrechungsblock aktiviert wurden.
- (5) Verzögerungsblöcke, die durch einen vorhergehenden Unterbrechungsblock aktiviert wurden.



Verzweigungen



■ Prioritäten

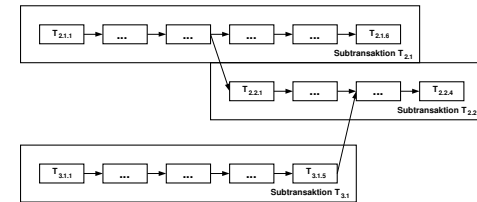
- Priorität des Vorgängers richtet sich nach den Nachfolgern
- $\text{Prio}(\text{Vorgänger}) = \max \text{Prio}(\text{Nachfolger})$

■ Antwortzeitanalyse

- Vorgänger: erweiterte Antwortzeitanalyse
- Nachfolger: erweiterte Antwortzeitanalyse für alle Nachfolger ausgehend vom Vorgänger



ODER-Abhängigkeiten



■ Antwortzeitanalyse

- für alle Vorgänger: erweiterte Antwortzeitanalyse
- Nachfolger wird für jeden Vorgänger analysiert

■ Achtung

- Aktivierungen des Nachfolgers
- ein Nachfolger muss mehrere Vorgänger behandeln



ODER-Abhängigkeiten

■ Prioritäten

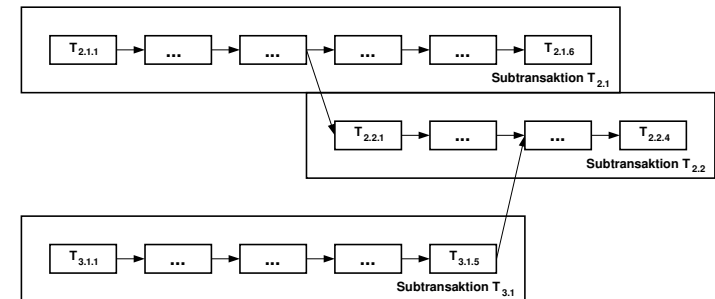
- mehrere Subtransaktionen *teilen* sich *denselben* Nachfolger
- diese Subtransaktionen können *verschiedene Deadlines* haben
- der Nachfolger müsste *verschiedene Prioritäten* besitzen

■ Lösung

- Priorität des Nachfolgers richtet sich nach den Vorgängern
- $\text{Prio}(\text{Nachfolger}) = \max \text{Prio}(\text{Vorgänger})$



UND-Abhängigkeiten



■ Antwortzeitanalyse

- für alle Vorgänger: erweiterte Antwortzeitanalyse
- Nachfolger wird für den *spätesten* Vorgänger analysiert



UND-Abhängigkeiten

■ Prioritäten

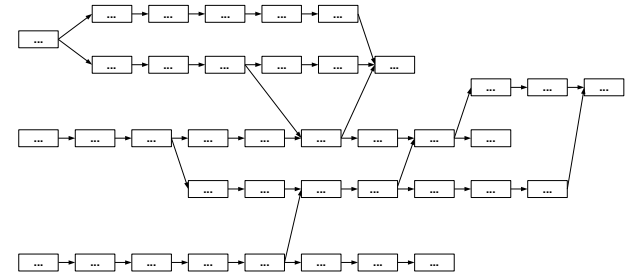
- alle Vorgänger haben einen *gemeinsamen Nachfolger*
- alle Vorgänger haben eine *gemeinsame Deadline*
- *innerhalb* der Vorgänger kann es auch *separate Deadlines* geben

■ Lösung

- Prioritäten der Vorgänger richten sich nach
 - der Priorität des Nachfolgers: $\text{Prio}(\text{Vorgänger}) = \text{Prio}(\text{Nachfolger})$
 - der separaten Deadline



Probleme - Prioritäten



■ Problematik: viele Subtransaktionen, viele Abhängigkeiten

→ hohe Prioritäten

→ evtl. Konflikte

→ nach Möglichkeit im Design vermeiden



Zusammenfassung

■ ProOSEK Abbildung

- Verzweigungen
- ODER-Abhängigkeiten
- UND-Abhängigkeiten
- aperiodische Ereignisse

■ ProOSEK: einfache Ablaufplanung

- Algorithmus
- trivialer Fall
- TASKs
- kritische Abschnitte
- ALARMe
- Self-Suspension
- beliebige Deadlines

■ ProOSEK: erweiterte Ablaufplanung

- Prinzip
- Verzweigungen
- ODER/UND-Abhängigkeiten



Ergebnis

■ voll ausformulierte Steuerung

- OSEK Konfiguration
- Implementierung der Anwendung
- Abbildung der Komponenten auf Ablauftabellen

■ Antwortzeitanalyse

