§ 4 Echtzeit-Programmierung

Kapitel 4 - Lernziele

- Wissen, was man unter Echtzeit-Programmierung versteht
- Die Forderungen bei der Echtzeit-Programmierung kennen
- Zwischen harter und weicher Echtzeit unterscheiden können
- Wissen, was man unter synchroner Programmierung versteht
- Die asynchrone Programmierung kennen
- Erklären können, was Rechenprozesse sind
- Wissen, welche Möglichkeiten zur zeitlichen Synchronisierung es gibt
- Semaphorvariablen anwenden können
- Verstanden haben, was Scheduling-Verfahren sind
- Die unterschiedlichen Scheduling-Verfahren anwenden können
- Wissen, was ein Schedulability-Test ist

§ 4 Echtzeit-Programmierung

Was heißt Echtzeit-Programmierung?

Nicht-Echtzeit-Datenverarbeitung

- Richtigkeit des Ergebnisses

Echtzeit-Datenverarbeitung

- Richtigkeit des Ergebnisses
- Rechtzeitigkeit des Ergebnisses

nicht zu früh, nicht zu spät

NICHT-ECHTZEIT-DATENVERARBEITUNG:

ECHTZEIT-DATENVERARBEITUNG:

Echtzeitprogrammierung (Realzeit-Programmierung)

– Erstellung von Programmen so, dass bei der Datenverarbeitung im Computer die zeitlichen Anforderungen an die Erfassung der Eingabedaten, an die Verarbeitung im Computer und an die Ausgabe der Ausgabedaten erfüllt werden.

Zeitanforderungen

– Zeitliche Anforderungen abhängig von den zeitlichen Abläufen im technischen Prozess

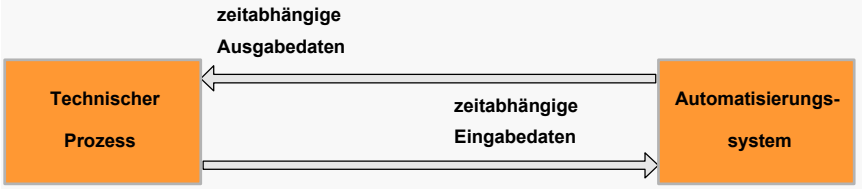
Koordination mit dem technischen Prozess

Echtzeit: den echten Zeitabläufen entsprechend, keine Zeitdehnung, keine Zeitraffung

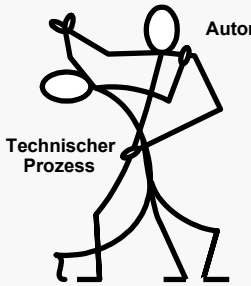
Arten von Anforderungen an das zeitliche Verhalten der Datenverarbeitung

- Forderungen nach Rechtzeitigkeit
- Forderungen nach Gleichzeitigkeit

Zusammenspiel technischer Prozess und AT-System



Bsp:



Fehlfunktion bei zu früh/zu spät ankommenden Sensor-/ Aktorwerten!

Unterschiede zwischen Informations-und Echtzeitsystemen

Informationssysteme	Echtzeitsysteme
datengesteuert	ereignis-/zeitgesteuert
komplexe Datenstrukturen	einfache Datenstrukturen
große Menge an Eingabedaten	kleine Menge an Eingabedaten
I / O -intensiv	rechenintensiv
hardwareunabhängig	hardwareabhängig

Wichtige Begriffe (1)

Reaktive Systeme

Echtzeitsysteme, die auf Eingabesignale vom technischen Prozess reagieren und Ausgabesignale zur Beeinflussung des technischen Prozesses liefern.

Bsp.: Automatisierungssysteme

Eingebettete Systeme

Integration des Automatisierungssystems in den technischen Prozess physikalisch und logisch.

Bsp.: Rasierapparat, Handy, Waschmaschine, Werkzeugmaschine

Wichtige Begriffe (2)

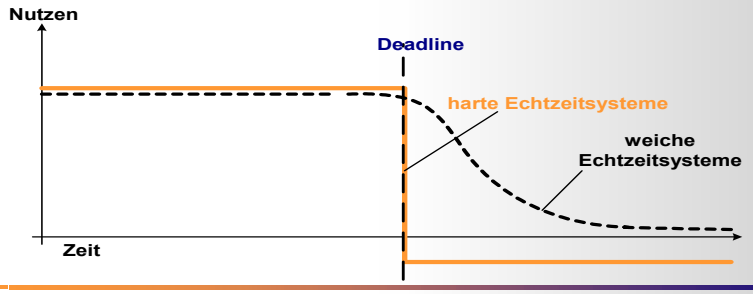
Harte Echtzeitsysteme

Einhaltung von strengen Zeitschranken (Deadlines) ist unabdingbar

Bsp.: DDC-Regelungen in Flugzeugen, Motorsteuerungen im Kraftfahrzeug

Weiche Echtzeitsysteme

Systeme bei denen eine Verletzung von Zeitschranken toleriert werden kann.



Forderung nach Rechtzeitigkeit

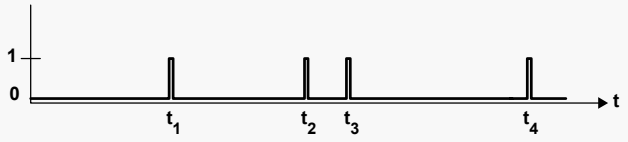
- rechtzeitiges Abrufen der Eingabedaten
- rechtzeitige Durchführung der Verarbeitung
- rechtzeitige Ausgabe der Ausgabedaten

Zeitbedingungen im Zusammenhang mit Rechtzeitigkeit:

- **Absolutzeitbedingungen**
  Bsp.: 11:45 Signal zur Abfahrt
- **Relativzeitbedingungen**
  Bsp.: wenn ein Messwert einen Grenzwert überschreitet, nach 10 Sekunden Abschaltsignal

Klassifizierung von Zeitbedingungen (1)

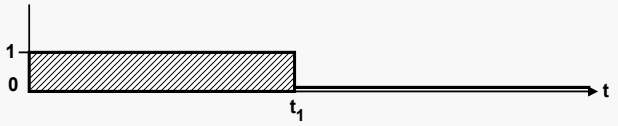
– Ausführung einer Funktion zu festen Zeitpunkten t_1, t_2, t_3



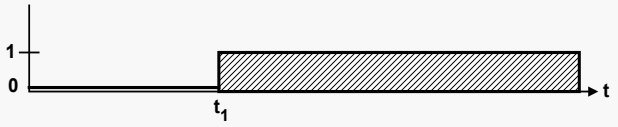
– Ausführung einer Funktion in einem zu jedem Zeitpunkt t_1 zugeordneten Toleranzintervall

Klassifizierung von Zeitbedingungen (2)

– Ausführung einer Funktion in einem Zeitintervall bis zu einem spätesten Zeitpunkt t_1



– Ausführung einer Funktion in einem Zeitintervall von einem frühesten Zeitpunkt t_1 an



Typische Anwendungsbeispiele mit Zeitbedingungen

	Absolutzeit-Bedingungen	**Relativzeit-Bedingungen**
Ausführung einer Funktion zu festen Zeitpunkten	Kennfeldaufnahme an Prüfständen	Analyse von Stoffen in der Chemie
Ausführung einer Funktion in einem Tolernaz-Zeit-intervall	Erfassung von Regelgrößen	Messwertüberwachung auf gleitende Grenzen
Ausführung einer Funktion in einem Zeitintervall bis zu einem spätesten Zeitpunkt	Erfassung von Datentelegrammen	Erfassung von Stückgut-Kennungen
Ausführung einer Funktion in einem Zeitintervall von einem frühesten Zeitpunkt an	Folgesteuerung bei Chargenprozessen	Erfassung von Signalen einer Lichtschranke

Forderung nach Gleichzeitigkeit

Vorgänge in Umwelt laufen gleichzeitig ab

- ➡ Echtzeit-Systeme müssen darauf “gleichzeitig” reagieren
- ➡ mehrere Datenverarbeitungsaufgaben müssen gleichzeitig durchgeführt werden

Bsp.:

- Reaktion auf gleichzeitige Fahrt mehrerer Züge
- Verarbeitung mehrerer gleichzeitig anfallender Messwerte bei einer Heizung
- Motorsteuerung und ABS-System gleichzeitig

Realisierung von Gleichzeitigkeit

- jede Datenverarbeitungsaufgabe durch getrennte Computer

echt parallel

- einen Computer für alle Datenverarbeitungsaufgaben

quasi gleichzeitig / quasi parallel

Voraussetzung:

Vorgänge in der Umwelt langsam im Vergleich zum Ablauf der Programme im Rechner

Forderung nach Determiniertheit

Determiniertheit = Vorhersehbarkeit des Systemverhaltens

Parallele bzw. quasi parallele Abläufe sind im allgemeinen nicht vorhersehbar, zeitliche Verschiebungen können zu unterschiedlichen Abläufen führen.

– das zeitliche Verhalten ist nicht determiniert

– keine absolute Garantie der Sicherheit von Automatisierungssystemen

Determiniertheit von Echtzeit-Systemen

Ein Echtzeit-System heißt determiniert, wenn es für jeden möglichen Zustand und für jede Menge an Eingabeinformationen eine eindeutige Menge von Ausgabeinformationen und einen eindeutigen nächsten Zustand gibt.

Voraussetzung: Endliche Menge von Systemzuständen!

Zeitlich determiniertes System:
Antwortzeit für alle Ausgabeinformationen ist bekannt!

In einem determinierten System ist garantiert, dass das System zu jedem Zeitpunkt reagieren kann, in einem zeitlich determinierten System zusätzlich auch bis wann die Reaktion erfolgt sein wird.

Dialogsysteme

- Eingabedaten über entsprechende Eingabemedien
  - Tastatur
  - Lichtgriffe
  - Maus
- Warten auf Antwort, d.h. Ausgabe von Ergebnissen auf einem Ausgabemedium
  - Bildschirm
  - Drucker

Bsp:
- Platzbuchungssysteme Luftfahrtgesellschaften
- Kontoführungssysteme bei Banken
- Lagerhaltungssysteme

Rechtzeitigkeit bei Dialogsystemen

zulässige Antwortzeit im Sekundenbereich

Automatisierungssysteme

Zeitliche Reaktion abhängig von den Vorgängen im technischen Prozess

Rechtzeitigkeit bei Automatisierungssystemen

zulässige Reaktionszeit im Bereich von Millisekunden/ Mikrosekunden

Methoden der Echtzeit-Programmierung ähnlich für

- Automatisierungssysteme
- Dialogsysteme

§ 4 Echtzeit-Programmierung

Arten des Vorgehens

– Synchrone Programmierung:
Planung des zeitlichen Ablaufs vor der Ausführung der Programme

Planwirtschaft

– Asynchrone Programmierung (Parallelprogrammierung):
Organisation des zeitlichen Ablaufs während der Ausführung der Programme

Marktwirtschaft

Bsp.: Zahnarztpraxis als Realzeitsystem (Tafelanschrieb)

Patient	Automatisierungsprogramm
Behandlungsstuhl und Zahnarzt	Computer, der Programme ausführt
Organisator Sprechstundenhilfe	Betriebssystem
Planung des zeitlichen Ablaufs **vor** der Behandlung mit Hilfe eines Terminkalenders	Synchrone Programmierung
Organisation des zeitlichen Ablaufs **während** der Behandlung (Aufruf aus dem Wartezimmer)	Asynchrone Programmierung, Parallelprogrammierung

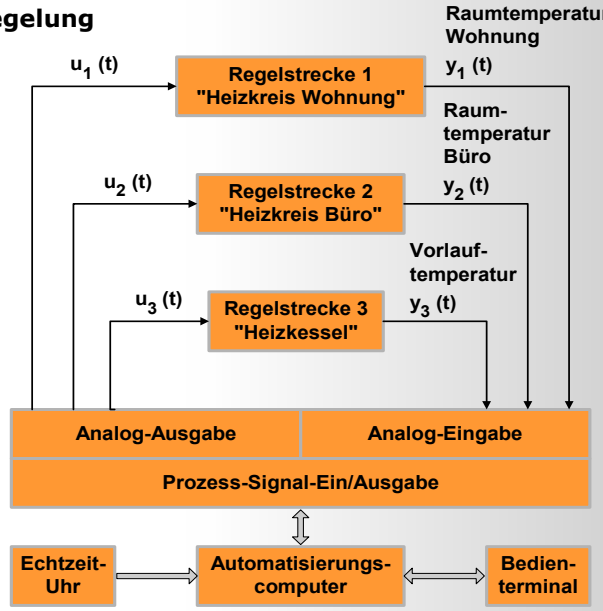
Verfahren der synchronen Programmierung

Synchrone Programmierung:
Planung des zeitlichen Verhaltens zyklisch auszuführender Teilprogramme vor der Ausführung

- Synchronisierung der zyklisch auszuführenden Teilprogramme mit einem Zeitraster
- Zeitraster über Echtzeit-Uhr, Unterbrechungssignal zum Aufruf über Teilprogramme

  Time triggered
- fest vorgegebene Reihenfolge des Ablaufs der Teilprogramme

Bsp.: Heizungsregelung



Fachtechnische (regelungstechnische) Konzeption

Planung

– Auslegung der Regelalgorithmen und Reglerparameter

– Festlegung der Abtastzeiten für die Regelkreise

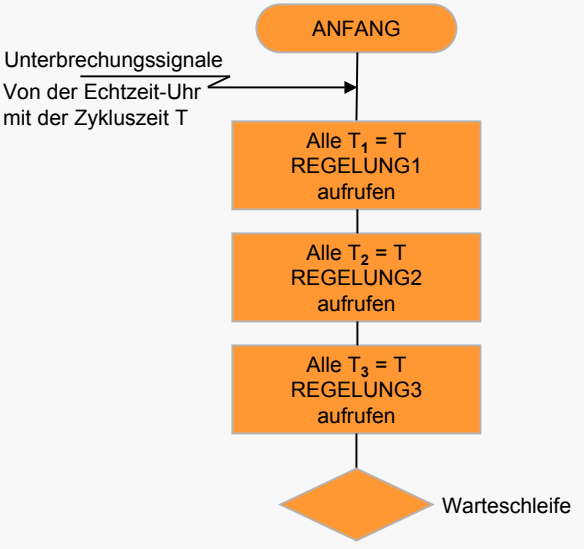
T	=	Echtzeituhr - Taktdauer
T_i	=	Abtastzeit für Regelkreis i
T_1	=	T
T_2	=	2T
T_3	=	5T

Zuordnung von Bezeichnern und Abtastzeiten

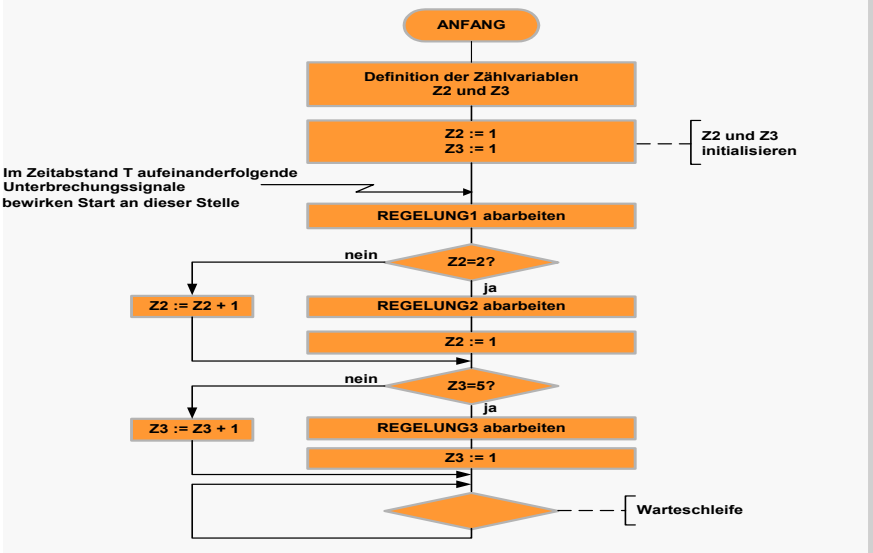
ein Teilprogramm für jeden Regelkreis

Teilprogramm	Bezeichner (Name)	Abtastzeit (Zykluszeit)
Temperatur-Regelung für die Regelstrecke „Heizkreis Wohnung“	REGELUNG 1	$T_1 = T$
Temperatur-Regelung für die Regelstrecke „Heizkreis Büro“	REGELUNG 2	$T_2 = 2T$
Temperatur-Regelung für die Regelstrecke „Heizkessel“	REGELUNG 3	$T_3 = 5T$

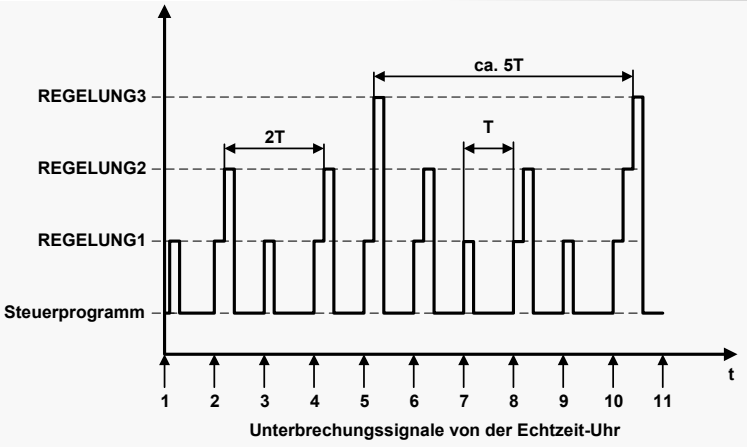
Grobentwurf bei synchroner Programmierung



Feinentwurf bei synchroner Programmierung



Zeitlicher Ablauf synchroner Programmierung



Annahme:
- Rechenzeit für Teilprogramme gleich groß
- Summe der Rechenzeiten der drei Teilprogramme kleiner als Zykluszeit

Eigenschaften der synchronen Programmierung (1)

- Forderung nach Rechtzeitigkeit wird näherungsweise erfüllt

  leichte Verschiebung

- Forderung nach Gleichzeitigkeit wird erfüllt, wenn Zykluszeit T klein gegenüber den Zeitabläufen im technischen Prozess

- Synchrone Programmierung gut für Echtzeit-Systeme mit zyklischen Programmabläufen

  vorhersehbares Verhalten

- Synchrone Programmierung ungeeignet für die Reaktion auf zeitlich nicht vorhersehbare (asynchrone) Ereignisse
  - Erhöhung der Rechenzeit durch ständiges Abfragen
  - Verzögerung der Reaktion

Eigenschaften der synchronen Programmierung (2)

- im Normalfall deterministisches Verhalten

- kein komplexes Organisationsprogramm

- etwas aufwendigere Planung (Entwicklung)

Nachteil der synchronen Programmierung

Änderung der Aufgabenstellung bedeutet Änderung der gesamten Programmstruktur !

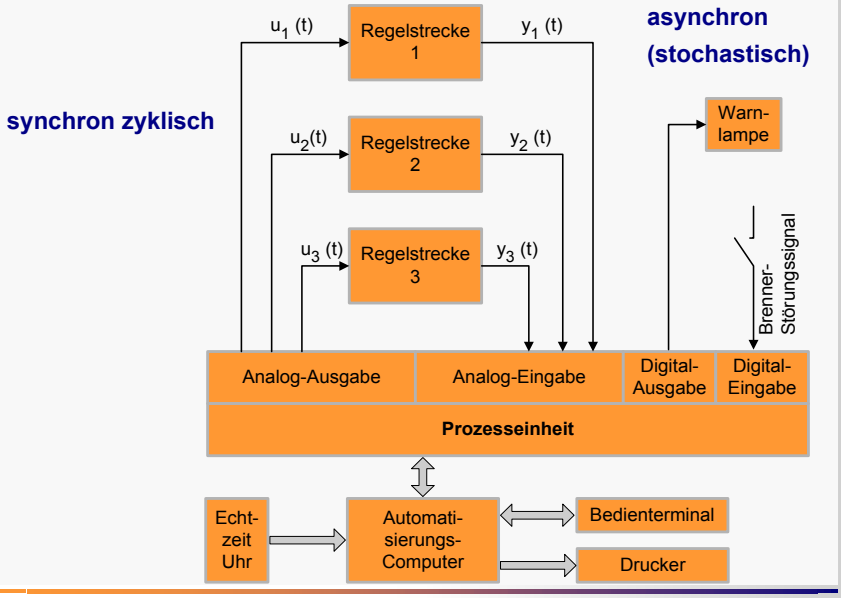
Bsp.: SPS-Programmierung

Verfahren der asynchronen Programmierung (Parallelprogrammierung)

Organisationsprogramm (Echtzeitbetriebssystem) steuert während des Ablaufs der Teilprogramme den zeitlichen Aufruf

- Aufruf der Teilprogramme, wenn Zeitbedingungen erfüllt sind
- Gleichzeitige Ausführung wird nach bestimmter Strategie sequenzialisiert
  - Zuordnung von Prioritätsnummern
  - Priorität umso höher, je niedriger die Prioritätsnummer

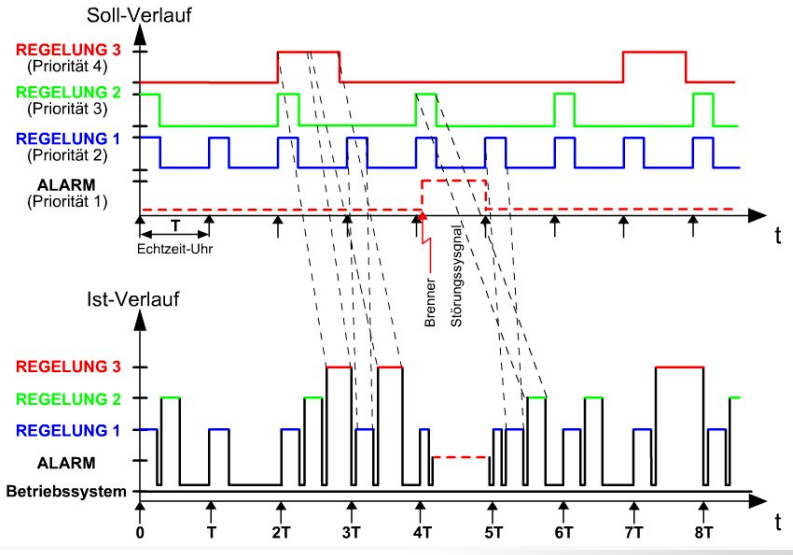
Bsp.: Heizungsanlage



Zuordnung von Bezeichnern, Abtastzeiten und Prioritäten

Teilprogramm	Bezeichner	Abtastzeit	Prioritäts-nummer	Priorität
Reaktion auf Brennerstörung mit Alarmmeldung	ALARM	–	1	höchste
Temperatur-Regelung für Heizkreis 1	REGELUNG1	$T_1 = T$	2	zweit-höchste
Temperatur-Regelung für Heizkreis 2	REGELUNG2	$T_2 = 2T$	3	dritt-höchste
Temperatur-Regelung für Heizkreis 3	REGELUNG3	$T_3 = 5T$	4	niedrigste

Zeitlicher Verlauf der Teilprogramme



Eigenschaften der asynchronen Programmierung

- Forderung nach Rechtzeitigkeit nur näherungsweise erfüllt

  schlecht für niederpriore Teilprogramme

- Zeitbedingungen um so besser erfüllt, je höher die Priorität des jeweiligen Teilprogramms
- Ist-Zeitablauf kann sich gegenüber Soll-Zeitablauf stark verschieben

  Teilprogramme können sich gegenseitig überholen

- Aufeinanderfolge der Teilprogramme ist nicht deterministisch, sondern stellt sich dynamisch ein
- Bei Programmerstellung lässt sich nicht im Voraus angeben, welches Teilprogramm zu welchem Zeitpunkt ablaufen wird

  - **einfache Entwicklung**
  - **Komplexität im Verwaltungsprogramm**
  - **Programmablauf schwer durchschaubar**

Ereignisgesteuerte Architekturen — asynchrone Programmierung

- Alle Aktivitäten als Folge von Ereignissen
  - Aktivierung von Tasks
  - Senden von Nachrichten
- Unterstützung durch Echtzeitbetriebssysteme
- nicht-deterministisches Verhalten
- flexibel bezüglich Veränderungen

Zeitgesteuerte Architekturen — synchrone Programmierung

- Periodische Durchführung aller Tasks und Kommunikationsaktionen
- Abtastung von externen Zustandsgrößen zu festgelegten Zeitpunkten
- wenig flexibel bei Änderungen
- einfach analysierbar

SPS-Systeme sind zeitgesteuerte Echtzeit-Systeme

§ 4 Echtzeit-Programmierung

Unterscheidung

- Programm (Anzahl von Befehlen)
- Ausführung des Programms (einmalige Ausführung der Befehlsfolge)
- Rechenprozess

Aufruf von Unterprogrammen

- Ausführung des aufrufenden Programms wird unterbrochen
- Ausführung des Unterprogramms
- Fortsetzung des aufrufenden Programms

Aufruf eines Rechenprozesses

- gleichzeitige Ausführung des aufrufenden Programms und des aufgerufenen Rechenprozesses

Rechenprozess **= Task**

Ein Rechenprozess ist ein von einem Echtzeit-Betriebssystem gesteuerter Vorgang der Abarbeitung eines sequentiellen Programms.

Eigenschaften von Rechenprozessen

– Rechenprozess beginnt mit Eintrag in eine Liste des Echtzeit-Betriebssystems und endet mit dem Löschen aus dieser Liste

– Rechenprozess existiert nicht nur während der Ausführung der Befehle, sondern auch während geplanter oder erzwungener Wartezeiten

Unterschiede zwischen Task und Thread

Task	Thread
Besitzer von Betriebsmitteln	Kann außer Prozessor selbst keine Betriebsmittel besitzen, verfügt über alle Betriebsmittel der Task, der er angehört
Eigener Adressraum	Adressraum der Task, der er angehört **gemeinsamer Adressraum**
Enthält einen oder mehrere Threads	Element einer Task
Kommunikation über die Task-grenzen hinaus, bevorzugt über Botschaften	Kommunikation zwischen den Threads, bevorzugt über gleiche Daten

4 Grundzustände

– Zustand “laufend” (running)
  - das Teilprogramm ist in Bearbeitung

– Zustand “bereit” oder “ablaufwillig” (runable)
  - alle Zeitbedingungen für den Ablauf sind erfüllt
  - es fehlt der Start durch das Betriebssystem

d.h. die Ausführung

– Zustand “blockiert” (suspended)
  - Rechenprozess wartet auf den Eintritt eines Ereignisses
  - Wenn das Ereignis eingetreten ist, Übergang aus dem Zustand “blockiert” in den Zustand “bereit”

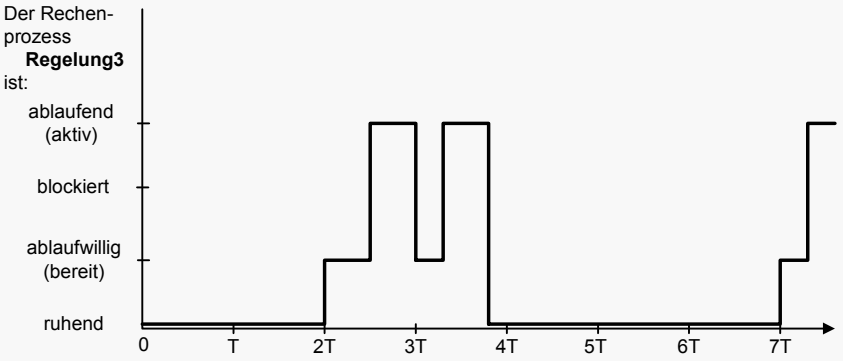
– Zustand “ruhend” (dormant)
  - Rechenprozess ist nicht ablaufbereit, weil Zeitbedingungen oder sonstige Voraussetzungen nicht erfüllt sind.

Zustandsdiagramm eines Rechenprozesses

Unter Einplanung versteht man
– die Beauftragung eines Rechenprozesses zyklisch oder zu bestimmten Zeiten

– der Übergang vom Zustand “ruhend” in den Zustand “bereit”

Verlauf des Rechenprozesses "Regelung3" beim Verfahren der asynchronen Programmierung



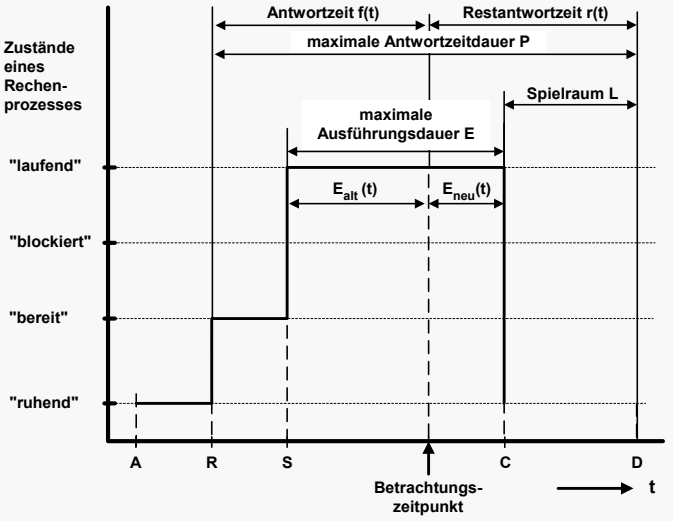
Prioritätsvergabe für Rechenprozesse

- statische Prioritätsvergabe
- dynamische Prioritätsvergabe (z.B. durch Verwendung von Deadlines)

Zeitparameter eines Rechenprozesses:

A:	Arrival time (Ankunftszeitpunkt)
R:	Request time (Einplanungszeitpunkt)
S:	Start time (Startzeit, Zuteilung eines Betriebsmittels)
C:	Completion time (Beendigungszeitpunkt)
D:	Deadline (Maximalzeit)
E:	Execution time (maximale Ausführungsdauer)
P:	Period time (maximale Antwortzeit)
L:	Laxity (Spielraum)
r:	Remaining flow time (Restantwortzeit)
f:	Flow time (Antwortzeit)

Auftreten der Zeitparameter eines Rechenprozesses



A: Arrival time
R: Request time
S: Start time
C: Completion time
D: Deadline

Zusammenhang zwischen den Zeitparametern

A <= R <= S <= C - E <= D - E

Maximale Ausführungsdauer (Rechenzeitdauer)

$E = E_{alt}(t) + E_{neu}(t)$

E_{alt} (t): bisherige Ausführungsdauer zum Betrachtungszeitpunkt

E_{neu} (t): verbleibende Ausführungsdauer

Zeitlicher Spielraum (laxity)

für die Ausführung eines Rechenprozesses
L = D - S - E

§ 4 Echtzeit-Programmierung

Klassifikation von Aktionen eines Rechenprozesses

- Zwei Aktionen in Rechenprozessen heißen parallel, wenn sie gleichzeitig ablaufen können
- Zwei Aktionen heißen sequentiell, wenn sie in einer bestimmten Reihenfolge angeordnet sind
- Zwei Aktionen aus zwei verschiedenen Rechenprozessen heißen nebenläufig, wenn sie gleichzeitig ablaufen können (äußere Parallelität)
- Zwei Aktionen eines Rechenprozesses heißen simultan, wenn sie gleichzeitig ausgeführt werden können (innere Parallelität)

Aktionen = Threads

Anforderung bei der Durchführung von Rechenprozessen

Synchronität zwischen Rechenprozessen und den Vorgängen im technischen Prozess.

Abhängigkeiten zwischen Rechenprozessen

– Logische Abhängigkeiten aufgrund der Vorgänge im technischen Prozess

Bsp.: Bevor die Sollwerte für die Regelung verwendet werden können, müssen sie mindestens einmal definiert werden.

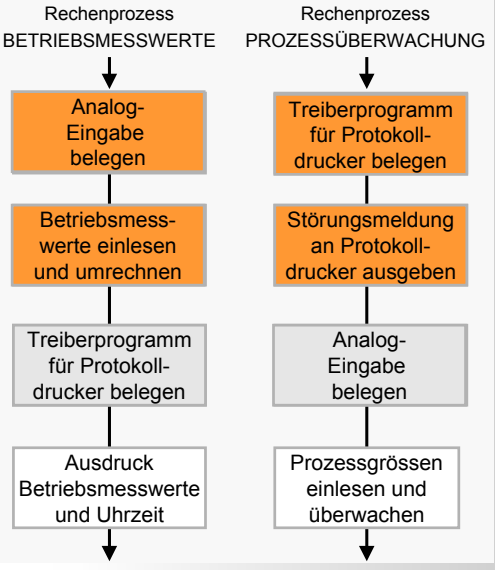
– Abhängigkeiten durch die gemeinsame Benutzung von Betriebsmitteln

Beispiel:

Abhängigkeit von Rechenprozessen aufgrund gemeinsam benutzter Betriebsmittel

Gemeinsame Betriebsmittel:
- Protokolldrucker
- Analog-Eingabe

Deadlock-möglichkeit!



Probleme durch Abhängigkeiten

Verklemmung (deadlock):

zwei oder mehrere Rechenprozesse blockieren sich gegenseitig

Permanente Blockierung (livelock, starvation)

Konspiration von Rechenprozessen blockiert einen Rechenprozess

➔ zeitliche Koordinierung der Rechenprozesse notwendig

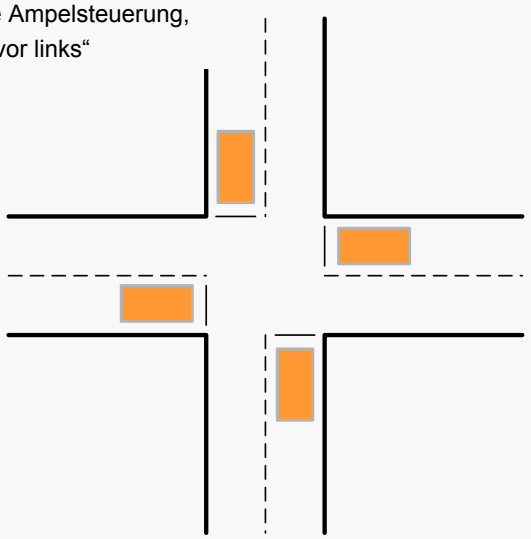
= Synchronisierung von Rechenprozessen

= Einschränkung des freien parallelen Ablaufs

Synchronisierung von Rechenprozessen ist gleichbedeutend mit Synchronisierung ihrer Aktionen

Beispiel für einen Deadlock

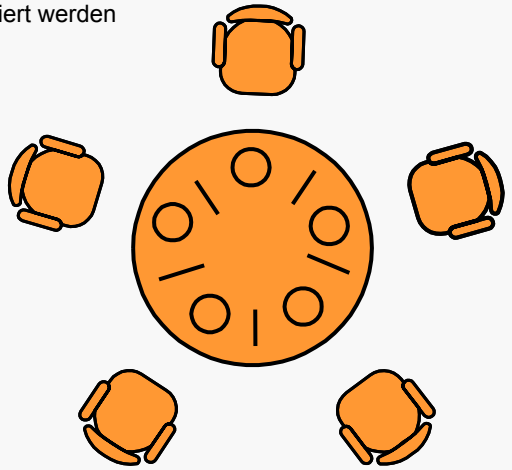
Straßenkreuzung ohne Ampelsteuerung,
Verkehrsregel „rechts vor links“



Beispiel für einen Livelock: „The Dining Philosophers"

- jeder Philosoph benötigt 2 Stäbchen zum Essen

- es kann kein Stäbchen reserviert werden



Hauptformen der Synchronisierung

Logische Synchronisierung

(aufgaben-/oder prozessorientierte Synchronisierung

– Anpassung des Ablaufs der Rechenprozesse an den Ablauf der Vorgänge im technischen Prozess

– Synchronisierung bedeutet

  - Erfüllung von Anforderungen bezüglich der Reihenfolge von Aktionen
  - Berücksichtigung vorgegebener Zeitpunkte bzw. Zeitabstände
  - Reaktionen auf Unterbrechungsmeldungen aus dem technischen Prozess

Betriebsmittelorientierte Synchronisierung

– Einhaltung von Bedingungen bezüglich der Verwendung gemeinsam benutzter Betriebsmittel (Ressourcen)

Synchronisierungsverfahren

- Semaphore
- kritische Regionen
- Rendez-vous-Konzept

Grundgedanke bei allen Synchronisierungsverfahren

- Rechenprozess muss warten, bis ein bestimmtes Signal bzw. Ereignis eintrifft
- Einfügen von Wartebedingungen an den kritischen Stellen

Semaphorkonzept

Synchronisierung von Rechenprozessen durch Signale (Dijkstra)

Semaphorvariable:
- positive, ganzzahlige Werte
- Semaphoroperationen: V(S) und P(S)

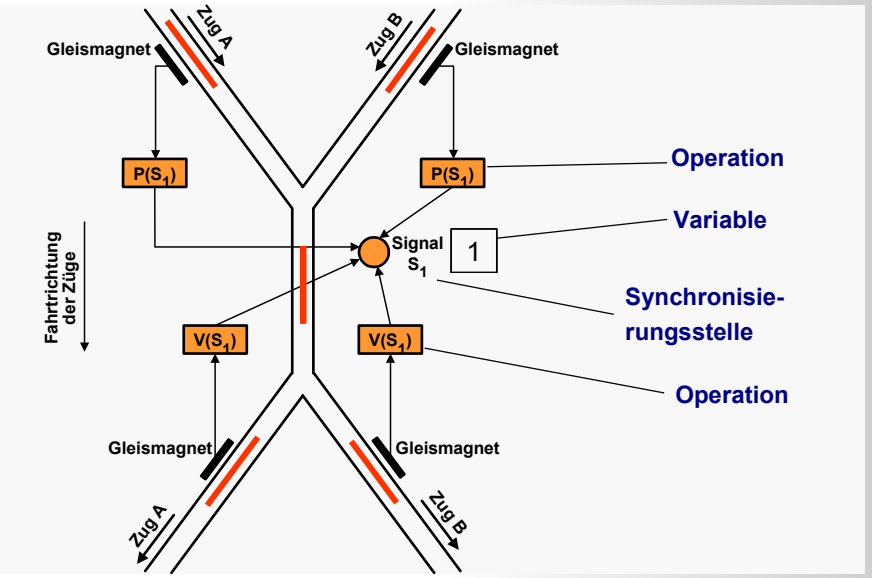
V(S_i): Operation V(S_i) erhöht den Wert der Semaphorvariable S_i um 1

P(Si): Operation P(S_i) bestimmt Wert von S_i

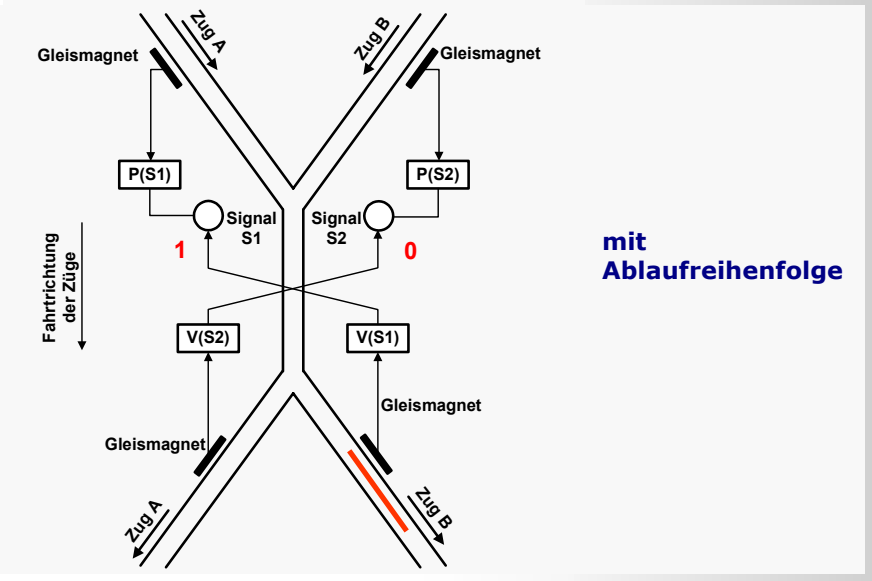
- ist Wert von S_i > 0
  wird S_i um 1 erniedrigt
- ist S_i = 0 wird gewartet,
  bis S_i > 0 geworden ist

unteilbar !

Beispiel 1: Zugverkehr über eine eingleisige Strecke



Beispiel 2: Zugverkehr über eine eingleisige Strecke

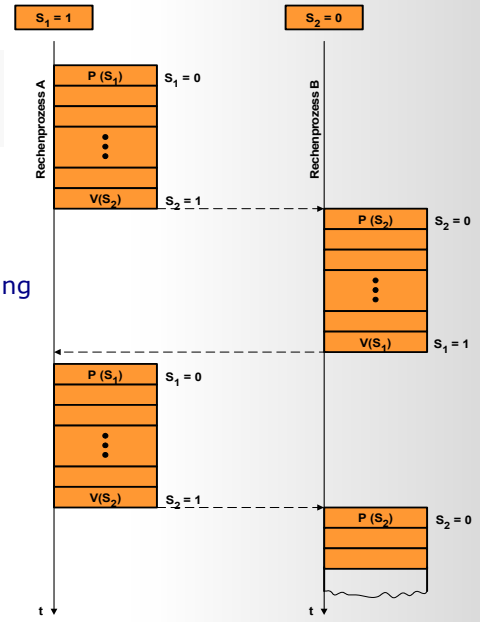


mit Ablaufreihenfolge

Beispiel:

Zwei Rechenprozesse sollen stets abwechselnd laufen.

logische Synchronisierung



§ 4 Echtzeit-Programmierung

Begriffsbestimmung

Synchronisierung =
Erfüllung zeitlicher und logischer Randbedingungen beim parallelen Ablauf von Rechenprozessen

Kommunikation =
Austausch von Daten zwischen parallel ablaufenden Rechenprozessen

Wechselwirkung: Synchronisierung - Kommunikation

Synchronisierung:	informationslose Kommunikation
Kommunikation:	Synchronisierung zum Informationsaustausch

Möglichkeiten der Datenkommunikation

- Gemeinsam benutzter Speicher (shared memory)
  - gemeinsame Variable
  - gemeinsame komplexe Datenstruktur
- Versenden von Nachrichten (messages)
  - Übertragung von Nachrichten von einem Rechenprozessor zu einem anderen (message passing)
  - vor allem bei verteilten Systemen

Arten der Kommunikation

– Synchrone Kommunikation

- sendender und empfangender Rechenprozess kommunizieren an einer definierten Stelle im Programmablauf

Warten durch Blockierung

– Asynchrone Kommunikation

- Daten werden gepuffert

keine Wartezeiten

§ 4 Echtzeit-Programmierung

Scheduling-Problem

- Rechenprozesse benötigen Ressourcen (Prozessor, Ein-/Ausgabegeräte usw.)
- Ressourcen sind knapp
- Rechenprozesse konkurrieren um Ressourcen
- Zuteilung der Ressourcen muss verwaltet werden

Beispiel: **Gleise im Bahnhofsbereich**

Scheduling

Unter Scheduling versteht man die Vergabe des Prozessors an ablaufbereite Rechenprozesse nach einem festgelegten Algorithmus (Scheduling-Verfahren).

Problem:

1. Gibt es für eine Menge von Tasks (Taskset) einen ausführbaren Plan (Schedule)?
2. Gibt es einen Algorithmus, der einen ausführbaren Schedule findet?

Beispiel: **Vorlesung - Raum - Zeit - Zuordnung**

Klassifizierung in Abhängigkeit vom Zeitpunkt der Planung

– Statisches Scheduling **unflexibel bei Änderungen**
  - Planung des zeitlichen Ablaufs der Tasks erfolgt vor der Ausführung (dispatching table)
  - Berücksichtigung von Informationen über Taskset, Deadlines, Ausführungszeiten, Reihenfolgebeziehungen, Ressourcen
  - Dispatcher macht Zuteilung gemäß dispatching table
  - Laufzeit- Overhead minimal
  - determiniertes Verhalten

– Dynamisches Scheduling **flexibel bei Änderungen**
  - Organisation des zeitlichen Ablaufs während der Ausführung der Tasks
  - erheblicher Laufzeit-Overhead

Klassifizierung nach Art der Durchführung

– **Preemptives Scheduling**
  - laufende Task kann unterbrochen werden
  - höherpriore Task verdrängt niederpriore Task

kooperatives Scheduling

– **Nicht-preemptives Scheduling**
  - laufende Task kann nicht unterbrochen werden
  - Prozessorfreigabe durch die Task selbst

Scheduling-Verfahren

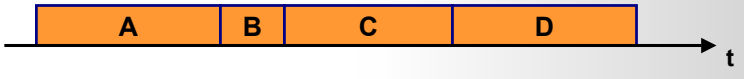
- FIFO-Scheduling (First-in-first-out)
- Zeitscheibenverfahren
- Scheduling mit festen (unveränderlichen) Prioritäten
- Ratenmonotones Scheduling
- Verfahren der kleinsten Restantwortzeit
- Verfahren des kleinsten Spielraums

FIFO - Scheduling

- Nicht-preemptives Scheduling
- Task, deren Einplanung am weitesten zurückliegt bekommt Prozessor
- einfache Realisierung

ungeeignet für harte Echtzeitsysteme

Tasks werden in der Reihenfolge ausgeführt, in der sie ablauffähig werden



Zeitscheibenverfahren (Round-Robin-Verfahren)

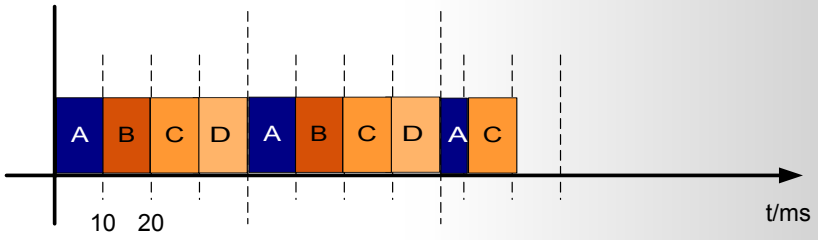
- Jede Task bekommt einen festgelegten Zeitschlitz, zu der sie den Prozessor bekommt
- Reihenfolge wird statisch festgelegt
- Abarbeitung einer Task “Stück für Stück”
- Verwendung bei Dialogsystemen (Multi-Tasking-Systeme)

für harte Echtzeitsysteme untauglich

Beispiel: Round-Robin-Verfahren

Jeder Zeitschlitz sei 10ms groß und die Tasks werden in der zyklischen Reihenfolge A-B-C-D abgearbeitet.
Zeitdauern der Tasks:

Task A:	25ms
Task B:	20ms
Task C:	30ms
Task D:	20ms



Scheduling mit festen Prioritäten

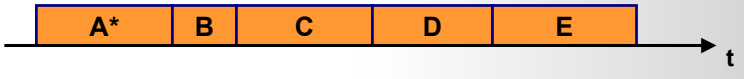
- Prioritäten werden statisch vergeben
- Task mit höchster Priorität bekommt Prozessor
- preemptive und nicht-preemptive Strategie ist möglich
- einfache Implementierung

für harte Echtzeitsysteme nur bedingt geeignet

Beispiel: Feste Prioritäten

Task	Prio	
A	1	höchste Prio
B	1	
C	2	
D	3	
E	3	

Je nach Strategie läuft eine der beiden Tasks A bzw. B so lange sie die höchste Priorität haben.



*** FIFO - Scheduling: A war zuerst ablauffähig**

Ratenmonotones Scheduling (Rate-Monotonic - Verfahren)

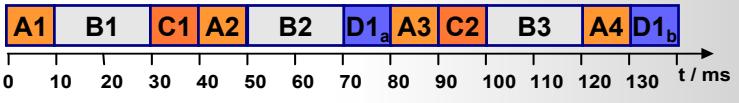
- Spezialfall des Verfahrens mit festen Prioritäten für zyklische Tasks
- Priorität um so höher, je kürzer die Zykluszeit (Periode) ist
- Task mit kleinster Zykluszeit erhält höchste Priorität
- preemptive Strategie

Verfahren wird in der Praxis häufig verwendet

Beispiel: Rate-Monotonic Scheduling

Task	Ausführungs-dauer	Zykluszeit	Priorität
A	10 ms	40 ms	1
B	20 ms	50 ms	2
C	10 ms	80 ms	3
D	20 ms	100 ms	4

Erste geplante Ausführung aller Tasks bei t = 0 ms.
Danach werden sollen Tasks zyklisch wiederholt werden.



Task D wird unterbrochen

Verfahren der kleinsten Restantwortzeit

= Earliest-Deadline-First-Verfahren

- Task mit kleinster Restantwortzeit bekommt den Prozessor
- preemptives Vorgehen
- hoher Rechenaufwand für das Scheduling
- Einhaltung von zeitlichen Anforderungen wird speziell unterstützt

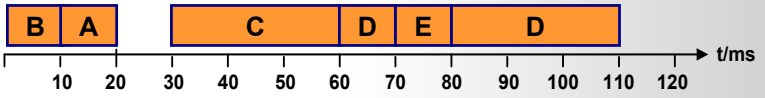
Beispiel: Earliest-Deadline-First-Verfahren

Task	Dauer	T_{min}	T_{max}
A	10 ms	0 ms	40 ms
B	10 ms	0 ms	30 ms
C	30 ms	30 ms	100 ms
D	40 ms	50 ms	200 ms
E	10 ms	70 ms	90 ms

t_{min}: frühester Termin

t_{max}: spätester Termin = Deadline

Ablauf:



Begründung: **Deadline von B ist früher als von A**
Deadline von E ist früher als von D
⇨ Unterbrechung

Verfahren des kleinsten Spielraums (least laxity)

- Task mit kleinstem Spielraum erhält Prozessor
- Berücksichtigung von Deadlines und Ausführungsdauern
- sehr aufwendiges Verfahren

für harte Echtzeitsysteme am besten geeignet

Ausführbarkeitstest (schedulability test)

Unter einem schedulability test versteht man die Prüfung, ob der zeitliche Ablauf für eine Menge von Tasks (Taskset) so geplant werden kann, dass jede Task ihre Deadline einhält.

Nachdem es sich dabei um eine mathematische Beweisführung handelt, muss unterschieden werden zwischen

- **notwendiger Bedingung**
- **notwendig und hinreichender Bedingung**

Theorem von Liu und Layland (Teil I)

Eine beliebige Menge periodischer Tasks (Deadline = Periode) kann durch das preemtive Earliest-Deadline-First-Verfahren ausgeführt werden, wenn gilt:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Wobei:

C_i : Ausführungszeit der Task i
T_i : Zykluszeit der Task i
n: Anzahl der Tasks

Bezüglich des EDF-Verfahrens ist dieser Test *notwendig und hinreichend*.

Für beliebige Schedulingverfahren ist dieses nur eine *notwendige* Bedingung.

Theorem von Liu und Layland (Teil II)

Eine beliebige Menge periodischer Tasks (Deadline = Periode) kann durch das Rate-Monotonic Verfahren ausgeführt werden, wenn gilt:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq U_n^* = n\left(2^{\frac{1}{n}} - 1\right), \quad n = 1, 2, \ldots$$

$U_1^* = 1{,}0$

$U_2^* = 0{,}828$

$U_3^* = 0{,}779$

$U_\infty^* = \ln 2 = 0{,}693$

Wobei:

C_i : Ausführungszeit der Task i
T_i : Zykluszeit der Task i
U_i: Utilization (Prozessorauslastung)
n: Anzahl der Tasks

und:

$$U_i = \frac{C_i}{T_i}$$

Hinreichender Test

Frage zu 4.1

Um welche Art von Echtzeitsystem handelt es sich bei den nachfolgend aufgeführten Systemen jeweils?

Antwort

	Hartes Echtzeitsystem	Weiches Echtzeitsystem
Zeitungsdruckwerk	X	
Steuerung eines elektrischen Fensterhebers	X	
Fernseher-Elektronenstrahlsteuerung		X
Telefonvermittlung		X
CNC-Fräskopfsteuerung	X	
Flugzeug-Triebwerkssteuerung	X	

Bei harten Echtzeitsystemen ist eine Verletzung der Zeitschranken mit einem Versagen des Automatisierungssystems gleichzusetzen. Dabei können Sach- oder Personenschäden auftreten.

Frage zu Kapitel 4.2

Welchen der folgenden Aussagen zum Thema „Echtzeitprogrammierung“ stimmen Sie zu?

Antwort

f ☐ Bei der Echtzeit-Programmierung steht nur der Zeitpunkt eines Ergebnisses im Vordergrund.

f ☐ Echtzeit bedeutet „so schnell wie möglich“.

f ☐ Bei weichen Echtzeit-Systemen müssen die Zeitschranken nicht eingehalten werden.

☑ Eine asynchrone Programmierung ist flexibler bei äußeren Einflüssen als eine synchrone Programmierung.

☑ Eine synchrone Programmierung setzt zyklische Programmabläufe voraus.

f ☐ Die synchrone Programmierung erfüllt nicht die Forderung nach Gleichzeitigkeit.

Frage zu Kapitel 4.4

a) In einem Automatisierungssystem greifen zwei Tasks auf den gleichen Temperatursensor zu.
b) Ein Regelalgorithmus wird in die drei Tasks „Istwert einlesen“, „Stellgröße berechnen“ und „Stellgröße ausgeben“ aufgeteilt, die immer in dieser Reihenfolge ablaufen müssen.

Um welche Art der Synchronisierung handelt es sich jeweils dabei ?
Wieviele Semaphorvariablen werden jeweils benötigt ?

Antwort

a) Es handelt sich um eine betriebsmittelorientierte Synchronisierung. Dabei wird für jedes Betriebsmittel eine Semaphorvariable benötigt (in diesem Beispiel eine).

b) Es handelt sich um eine logische Synchronisierung. Dabei wird für jeden Synchronisierungspunkt beim Übergang von einer Task zur nächsten eine Semaphorvariable benötigt, in diesem Beispiel also drei.

Frage zu Kapitel 4.6

Beim sogenannten „shortest-job-first“ Scheduling-Verfahren wird zur Laufzeit jeweils zuerst die Task bearbeitet, die die kürzeste Ausführungszeit hat. Laufende Tasks können dabei aber nicht von anderen unterbrochen werden.
Um welche Art von Scheduling-Verfahren handelt es sich?

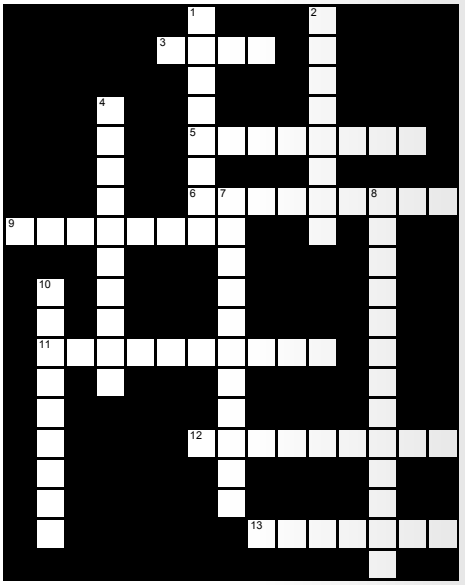
Antwort

Die Planung der Tasks erfolgt erst zur Laufzeit des Systems. Außerdem kann eine Task nicht eine andere unterbrechen.

Es handelt sich folglich um ein dynamisches, nicht-preemtives Scheduling-Verfahren.

Bemerkung:
Das „shortest-job-first“ Verfahren ermittelt immer einen Schedule, bei dem die mittlere Ausführungsdauer minimiert wird.

Kreuzworträtsel zu Kapitel 4



Kreuzworträtsel zu Kapitel 4

Waagerecht

3	Stapelverarbeitung in der Reihenfolge des Eintreffens (4)
5	Permanente Blockierung (8)
6	Organisation des zeitlichen Ablaufs während der Programmausführung (9)
9	Spätester Abarbeitungszeitpunkt einer Task (8)
11	Übergang vom Prozesszustand "ruhend" in den Zustand "bereit" (10)
12	Synchronierungs-Konstrukt (9)
13	Menge aller von einem Scheduler verwalteter Tasks (7)

Senkrecht

1	Bezeichnung für zwei Aktionen eines Rechenprozesses, die gleichzeitig ausgeführt werden können. (8)
2	Verklemmung (8)
4	Vergabe von Prozessorresourcen an ablaufbereite Prozesse (10)
7	Bez. für Aktionen, die in bestimmter Reihenfolge angeordnet sind (11)
8	Von Echtzeit-OS gesteuerter Vorgang der Abarbeitung eines sequenziellen Programms (13)
10	Unterbrechende Prozessorzuteilung (9)