Echtzeitsysteme

Fragenkatalog Ausarbeitung

Latest compile on:

26. März 2009

Inhaltsverzeichnis

1 The Real-Time Environment

1.1 Sprungantwort (Was ist das?) Kap 1 S 6

Die Sprungantwort ist charakterisiert durch das Verhalten des beobachteten Werts bei einer sprunghaften Ändern des kontrollierten Wertes.(zB durch öffnen eines Ventils) Ein wichtiger Parameter ist die rise time", sprich der Zeitintervall begrenzt durch die 2 Zeitpunkte bei denen 10 bzw 90% des Wertes erreicht wird. d^{object} und d^{rise} berechnet sich in dem man eine Gerade durch diese 2 Punkte legt und die Zeitpunkte betrachtet zu denen sie die stabilen Zustände schneidet.

1.2 Rise time (Bedeutung von rise time?) Kap 1 S 7

Unter Anstiegszeit und Abfallzeit (auch als englische Begriffe (rise time und fall time) gebräuchlich) versteht man in der Messtechnik die Zeit, die ein Pegelwechsel eines idealerweise rechteckförmigen Signals benötigt. Dazu wird die Zeit betrachtet, in der sich der Signalpegel zwischen zwei definierten Schwellen (häufig bei 10 % und 90 %) befindet. Häufige Anwendung finden die Begriffe in der (binären) Digitaltechnik. Dort beschreiben sie beim Umschaltvorgang die Zeiten, in denen das (analoge) Signal nicht mehr den alten und noch nicht den neuen definierten Logikpegel („0" bzw. „1") innehat („Flankensteilheit"). Betrachtet werden hierbei meist im Worst Case garantierte Zeiten, so dass sie die Zeit beschreiben, die ein Signal (beispielsweise in einem Computerprozessor) zum sicheren Umschalten zwischen den beiden binären Zuständen benötigt. (Das heißt, die als Maximalgrößen spezifizierten Anstiegs- und Abfallzeiten sind keine Messwerte, sondern ein Qualitätsmerkmal des betrachteten Bausteins bzw. der betrachteten Logikfamilie.)

1.3 Regelgrößen (Nenne charakteristische R.?) Kap 1 S 8

Symbol	Parameter	Sphere of Control	Relationship
d^{object}	controlled object delay	controlled object	physical process
d^{rise}	rise time of step response	controlled object	physical process
d^{sample}	sampling period	computer	$d^{sample} << d^{rise}$
$d^{computer}$	computer delay	computer	$d^{computer} < d^{sample}$
$\Delta d^{computer}$	jitter of the computer delay	computer	$\Delta d^{computer} < d^{computer}$
$d^{deadtime}$	dead time	computer and controlled object	$d^{computer} + d^{object}$

1.4 Regelstrecke, Zeitparameter (was gibt es für Z. für ine R.?)

is das ned das selbe wie grad oben?

1.5 OnLine und RT unterschiede Kap 1 S12

characteristic	hard real-time	soft real-time(online)
response time	hard-required	soft-desired
peak load performance	predictable	degraded
control of pace	environment	computer
safety	often critical	non-critical
size of data files	small/medium	large
redundancy type	active	checkpoint-recovery
data integrity	short term	long-term
error detection	autonomous	user assisted

2 Why a Distributed Solution

2.1 Data semantics and control strategy Kap 2 S 32

Sender/Receiver	Event Inf Ext.Control	Event Inf. Aut.Control	State Inf. Ext.Control	State Inf. Aut.Control
Event Information/External Control	Event Message	maybe	yes	yes
Event Information/Autonomous Control	no	maybe	yes	yes
State Information/External Control	yes	maybe	yes	yes
State Information/Autonomous Control	no	maybe	yes	State Message

3 Global Time

3.1 Accuracy def. Kap3 S 50

Die Abweichung (offset) einer Uhr k in Bezug auf die Referenzuhr z zu einem microtick i nennt man $accuracy_i^k$. Die maximale Abweichung über alle microticks i die von Interesse sind nennt man die $accuracy^k$ der Uhr k.

Wenn alle Uhren eines Ensembles extern mit einer Genauigkeit von A synchronisiert sind dann ist das Ensemble auch intern mit einer Genauigkeit von maximal $2A$ synchronisiert

Das Temporal Accuracy Interval ist die maximale Zeit zwischen der Beobachtung und Verwendung einer RT Entität.

3.2 Granularität / Precision (wie hängen sie zusammen?) Kap 3 S 48

Eine Uhr ist ein Gerät das einen Zähler beinhaltet und diesen periodisch nach den bestimmten Gesetzen der Physik (microticks) erhöht. Die Granularität einer Uhr k ist die Anzahl der Microticks der Referenzuhr (Dauer) zwischen allen zwei aufeinanderfolgenden Microticks von k.

Offset: gibt den microtick Abstand zwischen zwei Uhren (j, k) mit der selben Granularität an

$$offset_i^{jk} = |z(microtick_i^j) - z(microtick_i^k)|$$

Die Präzision ist der maximale Offset in einem Verbund von Uhren:

$$\Pi_i = \max_{\forall 1 \leq j,k \leq n} \{offset_i^{jk}\}$$

3.3 Potential Causality Kap 3 S 46

Wenn 2 Events auftreten kann man aus der zeitlichen Ordnung nicht unbedingt eine kausale Ordnung ableiten (Person betritt Raum - Telefon klingelt).

Wenn die zeitliche Ordnung zwischen Alarms hergestellt ist kann man alle definitiv später als der erste Alarm auftretenende Alarme als primären Alarm ausschließen.

3.4 Drift Rate (Wie groß ist die d. r. bei typische Uhren (Armbanduhr, PC Uhren)? Kap 3 S 49)

Der $drift$ einer Uhr k zwischen microtick i und microtick $i+1$ ist die Frequenzratio zwischen der Uhr k und der Referenzuhr zu der Instanz des microtick i. Berechnet wird diese durch Messen der Dauer eines Granulars der Uhr k bezogen auf die Referenzuhr dividiert durch die Anzahl n^k der Referenzuhrticks in einem Granular

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k}$$

Da der Drift einer guten Uhr nahe bei 1 liegt schreibt man die drift rate ρ_i^k als

$$drift_i^k = \mid \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} - 1 \mid$$

Die drift rate ist von Umgebungsvariablen abhängig. In spezifizierten Umgebungen kann die driftrate durch die *maximum drift rate* ρ_{max}^k angegeben werden.

Eine Uhr kann entweder einen inkorrekten Counter haben oder die drift Rate des Uhr liegt außerhalb der Spezifikation.

Bei typischen Uhren beträgt die Driftrate10^{-2} bis 10^{-7} sec/sec

3.5 Reasonableness Condition (Wann ist eine Zeit vernünftig?) Kap 3 S 52

Die globale Zeit t nennt man reasonable wenn alle lokalen Implementierungen der globalen Zeit die Bedingung $g > \Pi$ wobei $|z(microtick_i^j) - z(microtick_i^k) < \Pi$ erfüllen. g nennt man Granularität. Die reasonableness condition stellt sicher, dass der Synchonizationsfehler auf weniger als ein *macrogranule* (Dauer zw. 2 ticks der lokalen Uhr) begrenzt ist.

Wenn die reasonableness Condition erfüllt ist gilt für 2 unterschiedliche Uhren:

$$|t^j(e) - t^k(e)| \leq 1$$

Das bedeutet, dass die globale Timestamp eines events auf 2 unterschiedlichen Nodes maximal um einen Tick verschieden sein kann.

3.6 $\pi/\Delta - Precedence$ Kap 3 S 54

3 Nodes j,k und m. Jeder generiert Events. Alle events werden lokal zum selben global tick generiert und treten im Interval π auf wobei $\pi \leq \prod$ gilt. Events die in π auftreten dürfen nicht geordnet werden da sie als gleichzeitig auftretend angesehen werden. Wie

viele "granules of silence" müssen zwischen den events existieren damit man die zeitliche Ordnung immer bestimmen kann?

Ein Set von events E mit den 2 Dauern π und Δ wobei $\pi << \Delta$ so dass für beliebige 2 Elemente e_i und e_j dieses sets die folgende Bedingung hält:

$$[|\ z(e_i) - z(e_j)\ | < \pi] \vee [|\ z(e_i) - z(e_j)\ | > \Delta]$$

Nennt man $\pi/\Delta - Precedence$. Das bedeutet dass ein Subset von events die ca. zur selben Zeit passieren (nah beisammen innerhalb von π) durch einen Interval (mindestens Δ) von Elementen eines anderen Subsets getrennt sind. Wenn π 0 ist dann sind 2 events entweder gleichzeitig oder Dauer Δ auseinander.

3.7 Dense time Kap 3 S 56

Bei einer dense timebase können Events zu jedem Zeitpunkt auf der Zeitachse auftreten. Um die Ordnung zweier Events auf einer dense time base immer herstellen zu können müssen diese Events um mindestens 3 Granularitätseinheiten der globalen Zeit auseinander liegen. Der Grund dafür ist, dass die Uhren zu leicht unterschiedlichen Zeiten ticken und daher der zeitliche Unterschied (von 2 Ticks) von Events die nicht von allen Nodes direkt beobachtet werden zu ungenau ist um sie zu ordnen.

3.8 sparse time Kap 3 S 57

Wenn das Auftreten von "gleichzeitigen" Events auf einen aktiven Interval der Dauer ε begrenzt ist und das Auftreten 2er nicht gleichzeitigen Events nicht innerhalb der Dauer Δ liegen kann dann nennt man diese Zeitbasis *sparse* Bei einer spare time-base haben cluster keine untereinander synchronisierte Zeitbasis sondern nur lokal (lokal mit Granularität g).

Dabei darf ein empfangender Cluster keine zeitliche Ordnung von 2 Events ableiten wenn diese nicht nicht mindestens 1g/4g präzedent sind und weniger als 3 ticks auseinander liegen.

3.9 Agreement protokoll für Dense Time-base Kap 3 S 57

Damit alle Nodes eine konsistente Sicht der zeitlichen Ordnung haben müssen diese Informationen austauschen. Zuerst tauschen die Nodes die Informationen über deren Sicht der Welt aus (Also die Ordnung der Events). In einer zweiten Runde wird ein (deterministischer) Algorithmus durchgeführt damit jeder Node die selbe Sicht der Dinge hat.

Diese Protokolle haben den Nachteil dass sie eine zusätzliche Verzögerung einführen und Kommunikations- und Berechnungsresourcen benötigen.

3.10 Synchronisationsbedingungen (Was gibt es für S.?) Kap 3 S 59

Die Synchronisationsbedingung besagt, dass der Synchronisationsalgorithmus die Uhren so nah aneinander bringen muss damit sie im nächsten Resynchronisationsinterval den Präzessionsbereich nicht verlassen.

Ein Ensemble von Uhren kann nur dann synchronisiert werden wenn folgende *synchronization conditions* halten:
$\Phi + \Gamma \leq \Pi$.
$\Gamma = 2\rho R_{int}$
, also der Drift offset von der Länge des Resynchronisationsintervals R_{int} und der maximalen angegebenen Driftrate ρ abhängt.
Φ... Konvergenzfunktion, Γ... Drift offset, Π... Präzision.

3.11 Byzantinische Fehler (Was macht b. f. in fehlertoleranten Systemen?) Kap 3 S 60 und andere

Ein Byzantinischer Fehler ist ein "two faced" Verhalten. D.h der Fehler ist nicht konsident (Jeder Node sieht eine unterschiedliche Auswirkung des Fehlers). Z.B bei einer Uhr: Der fehlerhafte Node schickt Uhrzeit 07:50 an Node A und 08:10 an Node B

3.12 Central Master Algorithmus (Genauigkeit des CMA?) Kap 3 S 60

Ein einziger Knoten, der zentrale Master, sendet periodisch seinen Zeitwert in Form von Synchronisationsnachrichten an alle anderen Knoten, die Slave nodes. Sobald ein Slave den neuen Zeitwert vom Master empfangen hat, speichert er den Stand seines lokalen Zeitzählers als Nachrichtenempfangszeit. Die Differenz zwischen der Master-Zeit in der Synchronisationsnachricht und der gespeicherten Empfangszeit, korrigiert um die Verzögerung durch die Übertragung, ist ein Maß für die Abweichung der beiden Uhren. Der Slave korrigiert nun seine Uhr um die errechnete Abweichung, um sie mit der Master-Uhr in Einklang zu bringen. Die Präzision des Central Master Algorithm bekommt man aus der Synchronisationsbedingung: $\Pi_{central} = \varepsilon + \Gamma$, wobei ε der latency jitter, d.h. die Zeitdifferenz (Zeit lesen Master bis Empfang Slave) zwischen schnellster und langsamster Übertragung zu einem Slave, ist. Der Driftoffset ist $\Gamma = 2\rho R_{int}$ und beschreibt das driften der lokalen Uhr zwischen den Synchronisationszeiten.

Der Nachteil am CMA ist, dass es keine Korrektur bzw. Fehlererkennungsmöglichkeiten für Fehler im zentralen Master gibt.

Dies verwendet man meist während der startup Phase eines Verteilten Systems.

3.13 Fault Tolerant Average Kap 3

Fault Tolerance Average Ein System mit N Nodes wo k Byzantinische Faults toleriert werden sollen. (Es werden $N \geq (3k+1)$ Nodes benötigt) Alle Nodes schicken ihre Timestamp an alle anderen. Eine Konvergenzfunktion wird ausgeführt die den Fehler der Inkonsistenz aller Uhren eines Systems minimiert und k Byzantinische Fehler toleriert: Jeder Node misst dabei den Zeitunterschied zwischen seiner Uhr und den Uhrzeiten aller anderen Nodes. Die k größten und kleinsten Unterschiede werden entfernt. Von den restlichen $N-2k$ Unterschieden wird der Durchschnitt gebildet der dann der Korrekturwert für den Node ist.
Table for typical Jitter ε:

synchronization message assembled and interpreted	approximate range of jitter
at the application software level	$500\mu sec$ to 5 $msec$
in the kernel of the operating system	10 μsec to 100 μsec
in the hardware of the communication controller	less than $10\mu sec$

Das impossibility result gibt an dass es nicht möglich ist Uhren intern auf eine bessere Präzision als

$$\Pi = \varepsilon(1 - \frac{1}{N})$$

zu synchronisieren.

Die Präzision des FTA berechnet sich: Eine Uhr mit einem byzantinischen Fehler verursacht den Fehler:

$$E_{byz} = \Pi/(N-2k)$$

Eine Uhr mit k byzantinischen Fehler verursacht den Fehler:

$$E_{k_byz} = k\Pi/(N-2k)$$

Die Konvergenzfunktion ist dann gegeben durch:

$$\Phi(n,k,\varepsilon) = (k\Pi/(N-2k)) + \varepsilon$$

Die Präzision von FTA ist dann gegeben durch:

$$\Pi(N,k,\varepsilon,\Gamma) = (\varepsilon+\Gamma)\frac{N-2k}{N-3k} = (\varepsilon+\Gamma)\mu(N,k)$$

3.14 Konvergenzfunktion (Was ist das?) Kap 3 S 62

Die Konvergenzfunktion (bei einer Clock) gibt den offset der Zeitwerte genau nach der Resynchronization an. Die Uhren driften nach diesem Zeitpunkt wieder auseinander.

Drift offset Γ gibt die maximale Abweichung zw. 2 Uhren während des Resynchronizations Intervals R_{int} an. Dieser hängt von der Länge dieses Intervals und der drift rate ρ ab. $\Gamma = 2\rho R_{int}$

Wenn folgende Bedingung nicht erfüllt ist ist die Synchronization der Nodes unmöglich: $\Phi + \Gamma \leq \Pi$

FTA (fault tolerant average) Algorithmus ist ein Beispiel dafür. Ein system mit N Nodes wo k Byzantinische Faults toleriert werden sollen. Alle Nodes schicken ihre Timestamp an alle anderen. Jeder Node misst den Zeitunterschied zwischen seiner Uhr und allen anderen Nodes. Die k größten und kleinsten Unterschiede werden entfernt. Von den restlichen $N - 2k$ Unterschieden wird der Durchschnitt gebildet der dann der Korrekturwert für den Node ist.

Die Präzision berechnet sich mit:

$$\Pi(N, k, \epsilon, \Gamma) = (\epsilon + \Gamma)\frac{N - 2k}{N - 3k} = (\varepsilon + \Gamma)\mu(N, k)$$

3.15 Synchronisation (interne und externe Uhrensynchronisation? Messzahl für interne clock syn) Kap 3 S 59

Interne Uhrensynchronisation stellt sicher, dass die globalen Ticks aller funktionierenden Nodes innerhalb der Präzision Π liegen. Messgrößen sind die drift offset, Präzision

Driftoffset: $\Gamma = 2\rho R_{int}$ Die Präzision ist der maximale Offset in einem Verbund von Uhren:

$$\Pi_i = max\{offset_i^{jk}\}$$

Externe Uhrensynchronisation verbindet die globale Zeit eines Clusters mit einem externen Zeitstandard. (z.B durch GPS) Obwohl dadurch ein single point of error besteht ist dieser nicht so tragisch da durch reasonableness checks überprüft werden kann ob diese neue Zeitinformation korrekt ist. Wenn dem nicht so ist wird der Wert nicht übernommen und die Uhren driften auseinander haben aber keine komplett falsche Zeit

4 Modeling Real-Time Systems

4.1 Lasthypothese (Hochlast, max Ausführungszeit mit Task) Kap 4 S 72

Die Lasthypothese beschreibt die Antwortzeit eines Computersystems unterhalb einem spezifizierten Maximallast Szenario (peak load). Die Assumption coverage gibt die Wahrscheinlichkeit an dass dieses spezifiziertes Szenario eingehalten wird.

Um die maximale Ausführungszeit eines Tasks zu finden führt man die WCET Analyse durch. Die WCET hängt vom source code des tasks, den Eigenschaften des object codes und den Eigenschaften der Hardware ab. (S-Task)
Wenn der Task preemtive sein kann, muss außerdem der WCAO (Worst-Case Administrative Overhead) der durch den Context switch und dem Cache reloading entsteht berücksichtigt werden.

Um einen C-Task zu analyiseren hängt die WCET vom Verhalten aller Tasks ab ist also ein globales Problem. Methoden um die WCET eines C-Tasks zu analysieren sind:

- Experimentelle Daten der Implementierung finden (task, OS)
- Interaktion der Tasks minimieren und a priori Analyse der Kontrollstruktur (Anzahl der expliziten Synchronisationsaktionen die Context switches und OS services brauchen)
- Analyse von Subproblemen um Testcases zu finden
- Komplette Implementierung testen um die Assumption zu verifizieren und um safety margins zu ermitteln

4.2 Fehlerhypothese (Was ist das, Anzahl, Typ/Art, Annahmen über Unabhängigkeit) Kap 4 S 73

Eine fault Hypothese trifft eine Aussage über die Annahmen bezüglich der Arten und Häufigkeit/Frequenz von faults die das System tolerieren muss oder zeigt, dass diese nur selten auftreten.

Die Fault Hypothesis unterteilt Faults in 2 Gruppen:

- Level 1 faults: diese werden vom Fehlertoleranzmechanismus toleriert

- Level 2 faults: werden nicht toleriert, dürfen nur selten auftreten. Diese müssen *rare events* sein. Wenn während des Tests viele solcher Fehler auftreten liegt ein Fehler im Design nahe.

Die assumption coverage gibt an unter welchen Bedingungen die fault hypothese gilt (Fehler tritt alle xy h auf und recovery dauert jk msec)

Wenn beim Testen viele Level 2 faults auftreten ist entweder die fault-hypothese falsch oder die Implementierung hat Mängel.

Spezifikation der Fault Hypothese benötigt folgende Felder: Unit of Failure, Failure Mode, Frequency of Failure, Detection(how and when detected), State Recovery (how long)

4.3 Node integration, restart; States (h-State, ground-State) Kab 4/6

Node restart: self-test, dann Überprüfung des i-states durch Signatur der Datenstruktur. Node scant seine Instrumente und wartet auf einen cluster cycle um die aktuellen Informationen über die Umgebung zu bekommen. Node entscheidet über mode des kontrollierten Objektes um restart vektor(h-state output daten) herauszufinden welcher auf das Environment angewandt wird. H-state information(welche nicht durch I/O bestimmt sind) von extern holen und setzen. Danach ist der Node wieder Synchron mit dem rest des Clusters und kann seine Tasks starten,

Wenn der self-test erfolgreich ist dann war der Fehler nur transient.

h-state: Ist beim Start der Ausführung eines Tasks minimal und wächst bis zu einem gewissen Punkt der Ausführungszeit und schrumpft bis zur Terminierung wieder. Beinhaltet alle temporär benötigten Information für die Task Ausführung.
Ground state ist der Zustand wo kein Taks aktiv ist und keine Nachrichten unterwegs sind. Zu diesem Zeitpunkt ist die Reintegration eines Nodes leicht möglich.

4.4 h-state (definieren und plaudern, wieso/wie kann ein HW-fault zu falschen H-state führen?) Kap 4 S 76, 91

Ein h-state (history state) ist eine dynamische Datenstruktur eines Nodes der mit dem Fortschreiten der Berechnungen variert. Dieser wächst bis zu einem gewissen Zeitpunkt der Taskexecution und schrumpft dann wieder. Wenn ein Task unterbrochen wird müssen alle temporären Werte gesichert werden. Ein i-state dagegen ist eine statische Struktur die den Programmcode und Initialisierungsdaten in einem ROM hält.

Ein h-state kann in 3 Kategorien eingeteilt werden:

- (i): Input Daten die von den Instrumenten geholt werden.
- (ii): Output Daten die auf das Environment angewandt werden müssen.

- (iii): Restliche Daten. Diese müssen bei einem node restart von Node-externen Quellen geholt werden. Dieser Teil des h-States sollte möglichst minimal sein.

Ein HW-Fehler kann zu einem falschen h-state führen da durch diesen ein fehlerhafter Wert gespeichert und/oder weiterverarbeitet wird.

4.5 SRU (Was ist das, was ist ein Knoten) Kap 4 S 76

Eine SRU (smallest replaceable unit) ist die kleinste Einheit mit servicefähigen Schnittstellen (z.B. Steckverbindung im Vergleich zu nicht servicefähigen Verbindungen wie Lötverbindungen), die leicht ausgetauscht werden kann. Ein System besteht aus vielen solchen Einheiten, im Falle eines Fehlers können ganz einfach einzelne Einheiten durch neue ersetzt werden.

Ein Knoten (node) ist eine Hard-Software-Einheit, die aus folgendem besteht:

- Hardware: CPU, memory, real-time clock, process I/O, communication-interface
- Operating System: CPU Management, I/O support, error manager, com-support
- Initialization state: static data structure (app program code, init data (in ROM))
- History state: dynamic data structures mit Info über aktuelle und vergangene Berechnungen
- Ground state: minimaler History state wenn alle Tasks inaktiv und alle Kanäle gelöscht sind.

4.6 Komponenten (Interfaces) Kap 4 S 77

Ein Interface zwischen 2 Subsystemen kann man charakterisieren nach:

- control properties: Eigenschaften der Kontrollsignale (task aktivieren wenn Event auftritt)
- temporal properties: Zeitliche Beschränkungen die von control und data Signalen eingehalten werden müssen
- function intent: Die Spezifikation der beabsichtigten Funktion
- data properties: Struktur und Semantik von Datenelementen

Functional Intent ist eine Übermenge zu einer Funktion (z.B: Kontrolle von Abgaswerten, func: Je nach rechtlicher Lage) Es gibt das World Interface (zwischen System und Mensch) und das Message Interface(zwischen Systemen).

(ich glaub damit sind eigendlich die interfaces auf den folien gemeint)

Characteristik	Concrete World Interface	Abstract Message Interface
Information Representation	unique, determined by the given device	uniform within the whole cluster
Coupling	tight, determined by the specific I/O protocol of the connected device	weaker, determined by the message communication protocol
Coding	analog or digital, unique	digital, uniform codes
Time-base	dense	possibly sparse
Interconnection Pattern	one-to-one	one-to-many
Freedom in Design	determined by the format and timing of the physical I/O devices	determined by the uniform standards of the architecture

4.7 Kontrolle – Arten? Kap 4 S 82

Temporal control(time triggered), logical control (event triggered)

Logical control steuert den control flow innerhalb eines Tasks der durch die Programmstruktur und dem Input gegeben ist um die erwünschte Datentransformation durchzuführen

Temporal control kümmert sich darum festzustellen wann ein Task gestartet und blockiert werden muss. Letzteres ist der Fall wenn Zustände außerhalb des Tasks zu einem bestimmten Moment nicht gegeben sind.

5 Real-Time Entities and Images

5.1 Action delay Kap5

Die (maximale) Zeit zwischen dem Start einer Übertragung einer Message und dem Zeitpunkt wann diese Nachricht beim Empfänger permanent(also keine vorher gesendeten Nachrichten mehr ankommen können) wird nennt man action delay.

5.2 RT-Images (zeitliche Genauigkeit eines RT-Images, recent history) Kap 5 S 101ff

Ein RT image ist das momentane Abbild einer RT Entität. Ein RT image ist zu einem gewissen Zeitpunkt gültig wenn es eine präzise Repräsentation (in der Wert und Zeitdimension) einer zugehörigen RT Entität ist.

Eine RT Entität ist eine Zustandsvaribale und befindet sich innerhalb des Environments oder im Computersystem. (z.B.: Fluss einer Flüssigkeit in einem Rohr, Setpoint des control loops und die beabsichtigte Position des Kontrollventils). Eine RT Entität besteht aus statischen und dynamischen Attributen.

Ein RT Object entspricht einem Kontainer innerhalb eines verteilten Computersystems das RT images und RT entities hält. Jedem RT object ist eine RT Uhr zugewiesen.

Die zeitliche Genauigkeit ist definiert über die recent history der Beobachtung der entsprechenden RT Entität $\{t_i, t_{i-1}, t_{i-2}, t_{i-3}, ...t_{i-k}\}$ wobei $d_{acc} = z(t_i) - z(t_{i-k})$ der temporal accuracy interval oder auch, temporal accuracy, ist. $z(e)$ ist die Timestamp des events E das durch die globale Uhr generiert wird. Ein RT image ist temporally accurate wenn gilt:

$$\exists t_j \in RH_i : Value(RT\ Image\ at\ t_i) = Value(RT\ entity\ at\ t_j)$$

Die Verzögerung zwischen dem Beobachten der RT Entität und der Verwendung erzeugt einen Fehler, $error(t)$ der durch das Produkt des Gradienten des Wertes v der RT Entität multipliziert mit der Länge des Intervals zw Beobachtung und Verwendung:

$$error(t) = \frac{dv(t)}{dt}(z(t_{use}) - z(t_{obs}))$$

Wenn ein temporally valid RT image verwendet wird dann gilt folgende Formel:

$$error = (\max_{\forall t} \frac{dv(t)}{dt} d_{acc})$$

Damit ein Ergebnis zeitlich genau ist muss gelten:

$$z(t_{obs}) \leq z(t_{use}) \leq (z(t_{obs}) + d_{acc})$$

umgeformet ergibt das:

$$(z(t_{use}) - z(t_{obs})) \leq d_{acc}$$

5.3 Hidden Channel (Was ist das?) Kap 5 S 108

Ein hidden channel ist ein im Design nicht berücksichtigter Kanal über den Informationen fließen können. Dies bewirkt Probleme wenn der Informationsaustausch über diesen schneller ist als eine Nachricht zw 2 Nodes.

ZB: zw Sensor und Aktuator: Wenn eine Nachricht vom Operator über Öffnung eines Ventils von Operator zum Alarmmonitor länger braucht als Öffnen + Objektänderung + Sensorauslesen + Alarmsignal senden. (Alarm wird ausgelöst obwohl es kein Fehlverhalten ist) Um dieses Problem zu vermeiden muss der Empfänger warten bis er sicher ist dass keine Nachrichten die vorher gesendet wurden mehr ankommen können bevor er der Nachricht entsprechend handelt.

5.4 Permanente Nachricht (wann ist eine N p.?) Kap 5 S 108

Permanence is a relation between a given message Mi that has arrived at a Node N and all messages Mi-1, Mi-2, . . . that have been sent to this Node before (in the temporal order) message Mi. The message Mi becomes permanent at Node N as soon as all previously sent messages have arrived at N (or will never arrive). If actions are taken on non-permanent messages, then an inconsistent behavior may result.

5.5 Replika determinism (was ist das, was (zer)stört ihn?) Kap 5 S 111

Ist eine gewünschte Beziehung unter replizierten RT Objekten. Eine Gruppe von replizierten RT Objekten ist „replica determinant“, wenn alle Mitglieder dieser Gruppe nach außen den selben History- State haben und somit die selben Ausgangsnachrichten in dem gleichen Zeitabschnitten produzieren.

Replica Determinismus wird in erster Linie für zwei Dinge gebraucht:

- Die Testbarkeit der Gültigkeit eines Ergebnisses durch Mehrheitsentschluss
- Ermöglicht redundante Komponenten durch aktiv standby

Zerstört wird Replica Determinismus unter anderem durch:

- Inkonsistente Eingaben

- Inkonsistente Reihenfolge
- Nicht deterministische Programmkonstrukte
- Explizite Synchronisation (z.B. Wait)
- Unkontrollierter Zugriff auf globale Zeiten und Timeouts
- Dynamische preemptive "Scheduling Decisions"
- Konsistente Vergleichsprobleme (Software-Vielfältigkeit)

6 Fault Tolerance

6.1 Active Redundancy Components (Cost, fail-silent k+1, fail-consistent 2k+1, Byzantine 3k+1) Kap 6 S121

Die Anzahl an benötigten redundanten Komponenten welche bei Ausfall einer Komponente das Fortlaufen des Systems sichern ist abhängig von der Art des auftretenden Fehlers. Man unterscheidet hier zwischen consistent failure und inconsistent failure. Bei consistent failure ist der aufgetretende Fehler für alle Beobachter ident. Bei inconsistent failures kann dieser für die Beobachter verschieden sein. Inconsistent Failure sind die teuerstenFehler, da hier am meisten redundante Komponenten benötigt werden. Inconsistent Failures werden auch two-faced failures, malicious failures, oder Byzantine failures genannt. Bei den consistent failures kann man weiters fail silent failure und crash failure unterscheiden. Ein fail silent failure bedeutet dass der Node entweder nichts mehr sendet oder erkennbar (durch CRC) fehlerhafte Nachrichten zu korrekten Zeitpunkten versendet. Bei crash failure würde der Ausfall einer Komponenten den Absturz des gesamten Systems bedeuten.

Bei k auftetenden fehlerhaften Komponenten, benötigt man N redundante idente Komponenten:

- fail-silent: N=k+1
- fail-consistent: N=2k+1
- fail-malicious/byzantine: N=3k+1

6.2 Byzantinische Agreement (Was muss man für ein b. a. voraussetzen?) Kap 6 S121

Wenn man keinen Failure mode vorraussetzen kann benötigt man 4 Nodes für eine fault-tolerant unit (FTU) um einen Fehler erkennen zu können. (Da für das Erkennen eines byzantinischen Fehlers 3k+1 Nodes benötigt werden und dies der schwerwiegenste Fehlerfall ist)

Das Byzantinische Agreement Protokol benötigt um byzantinische Fehler von k Nodes erkennen zu können:

- FTU besteht aus $3k + 1$ Nodes
- Jeder Node ist mit den anderen Nodes der FTU durch $k + 1$ unabhängige Pfade verbunden

- Um den fehlerhaften Node erkennen zu können müssen $k + 1$ Kommunikationsrunden ablaufen. (Eine Runde: Jeder Node schickt eine Nachricht an alle anderen Nodes)
- Alle Nodes müssen mit einer bekannten Präzision synchronisiert sein

6.3 Error Detection in the Time Domain: Technique Kap 6

A priori wissen: Wenn man weiß, dass etwas zu einem fixen Zeitpunkt (eventuell mit jitter) gesendet werden muss, weiß man wenn ein Fehler aufgetreten ist da zu dem Zeitpunkt nichts passiert.

Oder durch referenz signal, fixe Verzögerung nachdem eine vorherige Nachricht empfangen wurde Wenn man die Regelmäßigkeit in der Aktivierung von Berechnungen kennt kann man feststellen wann eine Fehler in der Zeitdomäne aufgetreten ist.
(Verhindert babbling idiot)

(bin sehr unsicher ob das alles is)

6.4 Error Detection in the Value Domain: Technique Kap 6

Parity/CRC, a priori wissen (ASCI 1bit für parity feld weil nur 7 gebraucht werden), Mehrfachausführung von Tasks, Plausibilitätsprüfung, structural checks (assertions, robust data structures), trend analysis, TMR

(bin unsicher ob das alles is)

6.5 failure (Klassifizierung, aufzählen alle Punkte und erläutern, was consistent f?) Kap 6 S 120

Ein Failure ist eine Abweichung von einem spezifizierten Verhalten.

Man kann Failures einteilen in:

- Nature: Value, Timing
- Perception: Consistent(jeder sieht es gleich), Inconsistent(unterschiedliche Sicht auf unterschiedlichen Nodes)
- Effect: Benign(geringe auswirkung), Malign(katastrophe)
- Ofteness: Permanent, Transient

6.6 Fault/Failure - Ursachen Kap 6 S120

Ein Fault ist ein Event, es ist die Ursache eines Errors (unbeabsichtiger Zustand) der wenn er aktiviert wird (korrupte Daten gelesen) zu einem Failures führt.

6.7 Faults - Fehlerursachen(Klassifizierung) Kap 6 S124

Man kann Faults einteilen in:

- Nature: Chance, Intentional
- Perception: Physical, Design
- Boundaries: Internal, External
- Origin: Development, Operation
- Persistence: Transient, Permanent

6.8 Node integration, restart; States (h-State, ground-State) Kab 4/6

Node restart: self-test, dann Überprüfung des i-states durch Signatur der Datenstruktur. Node scant seine Instrumente und wartet auf einen cluster cycle um die aktuellen Informationen über die Umgebung zu bekommen. Node entscheidet über mode des kontrollierten Objektes um restart vektor herauszufinden welcher auf das Environment angewandt wird. H-state information von extern holen und setzen $->$ fertig

Wenn der self-test erfolgreich ist dann war der Fehler nur transient.

h-state: Ist beim Start der Ausführung eines Tasks minimal und wächst bis zu einem gewissen Punkt der Ausführungszeit und schrumpft bis zur Terminierung wieder. Beinhaltet alle temporär benötigten Information für die Task Ausführung.
Ground state ist der Zustand wo kein Task aktiv ist, der h-state minimal ist und keine Nachrichten unterwegs sind. Zu diesem Zeitpunkt ist die Reintegration eines Nodes leicht möglich.

6.9 Redundancy Arten und damit entdeckbare Fehler Kap 6 S 127

Type of Redundancy	Implementation	Type of Detected Errors
Time redundancy	The same software is exectuted on the same hardware during two different time intervals	Errors caused by transient physical faults in the hardware with a duration of less than one execution time slot
Hardware redundancy	The same software executes on two independent hardware channels	Errors caused by transient and permanent physical hardware faults
Diverse software on same hardware	Different software versions are executed on the same hardware during two different time intervals	Errors caused by independend software faults and transient physical faults in the hardware with the duration of less than one exection time slot
Diverse software on diverse hardware	Two different versions of the software are executed on two independent hardware channels	Errors caused by independent software faults and by transient and permanent physical hardware faults

7 Real-Time Communication

7.1 Flusskontrolle (Ziel, synchron Sender-Empfänger, explizit – implizit) Kap 7 S 150 & S 167

Das Ziel der Flusskontrolle ist es einen Informationsaustausch zwischen zwei Kommunikationspartner so zu kontrollieren, dass der Empfänger nicht überfordert wird. Man unterscheidet zwischen expliziter und impliziter Flusskontrolle.

Explizite Flusskontrolle: Der Sender sendet eine Nachricht an den Empfänger und wartet bis der Empfänger diese bestätigt (Acknowledge). Der Empfänger kann somit den Sender verlangsamen (back pressure flow control). Für die Error-Detection ist der Sender verantwortlich.

Implizite Flusskontrolle: Die maximale send rate wird vorher festgelegt. Kommunikationskannel ist unidirectional. Für die Error-Detection ist der Empfänger verantwortlich.

7.2 Bitweise Arbitration Kap 7 S161

Bitweise Arbitration benötigt man um den Nodes fairen Zugang zu dem Übertragungsmedium zu erlauben ohne das wichtige Nachrichten zu spät versendet werden.

Dazu hat, zB CAN, ein 11 bit Arbitrationsfeld im CAN Header. Eine 0 ist dabei dominant und eine 1 rezessiv. Wenn ein Node senden will schickt er die ersten Bits und falls 2 Nodes gleichzeitig senden werden so lange die bits des Datafields verglichen bis bei einem eine 0 und bei dem anderen eine 1 auftaucht. Der Node der die 1 übertragen hat bricht die Übertragung daraufhin ab.

Dabei ist es möglich Prioritäten zu vergeben da zB eine Nachricht mit einem Arbitrationsfeld mit 11 “0” ern immer gesendet werden kann.

7.3 PAR-Protokolle, flow - control, delays der Protokolle (allgemein) Kap 7 S 150

Explicit flow control: Ack von Empfänger, retry wenn ACK ausbleibt, timeout, max retry. Bidirektional delay existiert je nach timeout und Nachrichtenübertragungsdauer. Durch retrys kommt es zu protokoll jitter, da die meisten Nachrichten gleich korrekt ankommen manche aber erst beim 2ten oder 3ten mal usw.

Implicit flow control: a priori ausgemachte Zeitpunkte der Datenübertragung. Error detection beim Empfänger. Fault tolerance durch mehrfache Übertragung der Nachrichten. Unidirektional

7.4 Thrashing Kap 7 S 152

Thrashing bedeutet, dass der Durchsatz eines Systems bei zunehmender Last einbricht. Alle Mechanismen die eine überproportionale Zunahme von Resourcen bei Lasterhöhung benötigen können Thrashing verursachen.(z.B. Retry durch timeout bei PAR, Dynamischem Scheduling verursacht einen scheduling Overhead der mit Last zunimmt und dadurch weniger Resourcen für produktive Arbeit zur Verfügung stehen)

Saturation Point Punkt wo mehr Durchsatz theoretisch nichtmehr möglich ist
Thrashing Point: Punkt an dem der Durchsatz einbricht

7.5 CSMA/CD LON Kap 7 S 160

Carrier Sense Multiple Access/Collision Detection ist ein Protokol zur Steuerung des Medienzugriffes in einem verteilten System ohne Notwendigkeit für eine zentrale Kontrolle. Ethernet, LON verwendet dies.

Dabei wartet ein Node eine zufällige Zeitdauer bevor er zu senden anfängt. Die Zeitdauer die gewartet wird ist eine Funktion der Last auf dem Kanal um die Kollisionswahrscheinlichkeit unter Hochlast zu minimieren. Das nennt man stochastic back pressure flow control

7.6 ARINC 629 Kap 7 S 162

ARINC 629 ist ein Warteraum Protokoll und verwended das Minislotting Prinzip um auf ein shared Medium (in dem Fall ein Kanal) zuzugreifen. Es wird in Flugzeugen eingesetzt. Phase 1: Es existiert ein “Warteraum“ in den alle Prozesse rein gelassen werden die senden wollen. Nachdem eine gewisse Zeit (Synchronization Gap SG) abgelaufen ist dürfen alle Prozesse die in diesem verteilten Warteraum sind senden und haben dafür eine ”epoche“ Zeit.

Phase 2: Jeder Prozess hat einen terminal gap (TG) unterschiedlicher Länge ($TG < SG$). Sobald diese Zeit abgelaufen ist und der Kanal noch frei ist darf gesendet werden und der Transmit Interval (TI) der verhindert, dass ein Prozess den Kanal monopolisiert wird gestartet.

SG und TI sind ident für alle Prozesse. TG ist unterschiedlich für alle Prozesse. Es gelten

$$SG > \max_{\forall i} TG_i$$

und

$$TI > SG$$

Typische Werte für die Timeouts einen ARINC 629 Bus bei einer Datenrate von 2Mbit/s sind: 4-128µsec TG
SG: länger als der längste TG

Transmit interval: 0,5-64msec

7.7 Token Protokoll (Gibt es in Echtzeitsystemen mit t.p. Probleme?) Kap 7 S 161

Bei Tokenring erfolgt die Medienzugriffskontrolle durch eine spezielle Kontrollnachricht, Token genannt. Nur der Node der den Token hält darf senden, nach dem Senden gibt er den Token weiter. (z.B Profibus) Token-hold time THT (wie lang darf ein Node den Token halten) Token-rotation time (wie lang dauert es bis ein Node den Token wieder hat)

Problematisch ist wenn der Node der den Token besitzt abstürzt und erst ein anderer Node ihn generieren muss wenn dieser den Ausfall bemerkt. (Single Point of Failure) WCET Analyse ist ein globales Problem da der Ausfall eines Nodes (wenn er den Token hält) alle Nodes betrifft.

7.8 CAN – Buch (TTP; TDMA- Vor- und Nachteile) Kap 7 Seite 161, 163, 164,165

7.8.1 CAN (Control Area Network)

CAN (Control Area Network) ist ein Carrier Sense Multiple Access Collision Avoidance Protocol (CSMA/CA). Dabei wird von jedem Node wenn er Nachrichten senden will die Bits des Arbitration Feldes (11bit) auf den Kanal gelegt. Wenn 2 Nodes gleichzeitig senden wollen wird das Arbitration Feld so lange übertragen bis sich eine Stelle der 2 AFs unterscheiden und der Sender mit einem 1er an dieser Stelle bricht die Übertragung ab. Durch dieses Arbitration Feld lässt sich eine Nachrichtenpriorität festlegen.

Eine CAN Nachricht besteht aus 6 Feldern: Arbitration (11bit) Control (6bit) Data(0-64) CRC (16) A (2) EOF (7) Das CRC Feld deckt das Arbitration, Control und Data Feld ab.

CAN erlaubt externe Kontrolle, ist flexibel, erlaubt Sporadic Data, Spontane Services und bietet Probabilistic Access.
Nachteil: Probabilistic Access, nicht Replica deterministisch

7.8.2 TTP

Das TTP Protokol ist ein TDMA (Time Division Multiple Access) Protokoll. TDMA ist eine verteilte statische Mediumszzugriffsstrategie. Dabei wird das Zugriffsrecht auf den Kanal an den Ablauf der Zeit geknüpft. Dafür wird eine globale Zeitbasis benötigt. Bei TDMA wird jedem Node ein Slot zugewissen. Die Sequenz der Slots über alle Nodes

nennt man eine TDMA round. Ein Node kann nur eine Nachricht pro Runde verschicken. Wenn keine Daten gesendet werden müssen wird nur ein leeres Frame verschickt.

Das Time Triggered Protocol bietet:
Vorteil: einfache Composability, Error Detection und Protection, Regular Data, einfaches Interface und ist Replica deterministisch.
Nachteil: nicht flexibel, schlecht für sporadische Nachrichten mit baldiger Deadline
ET Protokolle sind schneller als TT Protokolle wenn viele sporadische Nachrichten zu unbekannten Zeitpunkten gesendet werden. Wenn viele periodische Daten verschickt werden ist ein TT Protokoll schneller.

(nachteile von CAN abgesehn von den standard ET nachteilen?)

7.9 CSMA/CD (Gibt es in Echtzeitsystemen mit c/c Probleme?)

Carrier Sense Multiple Access / Collision Detection protocoll , zB. Ethernet oder LON verwendet random Wait times um Kollisionen zu vermeiden.

Nicht deterministisch. Wenn es keine Prioritäten gibt kann es passieren, dass wichtige Nachrichten nicht schnell genug zugestellt werden können.

7.10 MFM / NRZ (Vorteile – Nachteile) Kap 7 S 167

7.10.1 NRZ: Non Return to Zero

NRZ: Non Return to Zero: 1 ist high, 0 ist low. Von diesem Code lässt sich das Taktsignal nicht regenerieren da es lange Folgen von high oder low Zuständen geben kann. Dazu muss bitstuffing eingesetzt werden. Vorteil: Wenige Signal transitions

7.10.2 MFM: Modified Frequency Modulation

MFM: Modified Frequency Modulation: 50:50 Datapoints und Clockpoints. 0 ist kein Signal change bei Datenpunkt und 1 ist signal change bei Datenpunkt. Eine Folge von mind 2 0ern benötigt einen signal chance zu einem Clockpunkt. Vorteil: Clockregenerierbar Nachteil: Halber Durchsatz da jede zweite Transition für die Clockpunkte "vergeudet" wird. Oder doppelte Frequenz

7.10.3 Manchester Code:

Manchester Code: 1 transition low-high 0 transition high-low Taktregenerierung möglich Halbe Feature size $->$ doppelte Frequenz oder halber Durchsatz, hohes EMI wegen höherer Flankensteilheit

8 The Time-Triggered Protocols

8.1 Wie funktioniert TTP?(S181ff) Wie funktioniert die Clock-Sync bei Hochstart und im Betrieb. (S185)

TTP ist ein TDMA (Time Division Multiple Access) Protokoll. Jeder Node hat eine MEDL (Message Descriptor List), welche den "Fahrplan" enthält. Jeder Node kann nur in den ihm zugewissenen Slots Daten senden. Aus der Abwesenheit einer erwarteten Nachricht kann auf den Ausfall eines Nodes geschlossen werden (Membership Service).

Die Clock-Synchronisierung erfolgt implizit. Da die Reihenfolge der Nachrichten a priori festgelegt wurde, ist bekannt wann sie ankommen sollen. Die Differenz zwischen erwarteter und realer Ankunfszeit ist ein Indikator für das Offset der Clocks.

9 Input/Output

9.1 Sampling/polling/interrupt – robust? Kap 9 S198

9.1.1 Sampling-Polling

Sampling bedeutet, dass der Zustand einer RT Entität periodisch (Diese Zeitpunkte nennt man sampling points) abgefragt wird. Dadurch bleibt die zeitliche Kontrolle immer innerhalb des Computer Systems. Den konstanten Zeitinterval zw. 2 benachbarten sampling Punkten nennt man sampling interval.

Beim Sampling muss wenn jedes Event der RT Entität wichtig ist ein Speicherelement implementiert werden das den Zustandswechsel bis zum nächsten Sampling Punkt speichert. (Dies liegt innerhalb des Sensors)

Der Unterschied zw Sampling und Polling besteht in der Position des Speicherelements (Bei Polling ist es innerhalb des Computers) Zw Sampling und Polling gibt es im Normalbetrieb keinen Unterschied allerdings ist Sampling im Fehlerfall robuster weil ein transienter Fehler auf der Leitung immer gespichert wird. Außerdem geht der Zustand des Speicherelements bei einem Neustart verloren.

9.1.2 Interrupts

Interrupts erlauben es einem System das außerhalb der Sphere of control des Computers ist zeitliche Kontrollabläufe innerhalb dessen zu beeinflussen. Wenn ein Zustandswechsel in einem Speicherelement auftritt und dieser Interrupt aktiviert ist zwingt ein Hardware Mechanismus einen Kontrolltransfer zu einer entsprechenden Interrupt Service Routine. Interrupts sind problematisch (sogar problematischer als Polling) da sie zu jedem beliebigen Zeitpunkt auftreten können und da jeder transiente Fehler auf einer Leitung die zeitlichen Kontrolle beeinflusst.

Das Speicherelement liegt bei Interrupts innerhalb des Computers

Interrupts werden dann benötigt wenn die Reaktionszeit auf ein Event kleiner als 1/10 der WCET ist. Um Probleme mit Interrupts zu vermeiden sollten Interrupts in Zeitbereichen bei denen keine Interrupts auftreten können deaktiviert werden.

10 Real-Time Operating Systems

10.1 The Non-Blocking Write Protocol (S217f)

NBW benötigt ein Concurrency Control Field (CCF). Zugriffe auf das CCF müssen atomic sein. Das CCF wird mit 0 initialisiert.

```
Writer:
start: CCF_old = CCF;
CCF = CCF_old+1;
<write data into structure>
CCF = CCF_old+2;

Reader:

start:  CCF_begin=CCF;
if(CCF_begin % 2 == 1) //odd
goto start;
<read data structure>
CCF_end = CCF;
if(CCF_end != CCF_begin) goto start;
```

11 Real-Time Scheduling

11.1 Schedulability Test (Was ist das? Komplexität? NP-hard?) Kap 11 S229

Ein Schedulability Test beurteilt ob ein Set von Tasks, die sich im ready Zustand befinden, scheduled werden können ohne dass einer seine Deadline verpasst. Es gibt *exact*, $sufficient$ und *necessary* schedulability tests.

Ein Scheduler ist optimal wenn ein exakter schedulability test die Existenz einer schedule findet. Dies ist ab einer einzigen shared Resource ein NP komplettes Problem. Sufficient/neccessary schedulability sind einfacher aber nicht optimal. Ein Task ist definitiv nicht schedulable wenn ein neccessary schedulability test negativ ausfällt. Ein Task ist definitiv schedulable wenn ein sufficient schedulability test positiv ausfällt.

11.2 Adversary Argument Kap 11 S229

Man kann zwischen periodischen (task request time ist a periori bekannt) und sporadischen Tasks (task request time ist nicht a periori bekannt nur der minimale Abstand zw. 2 Ausführungen) unterscheiden. Benötigte Parameter sind: Periode p_i, Execution Time c_i und deadline interval d_i. Die laxity l_i eines Tasks ist die Differenz zwischen d_i und c_i. Der deadline interval ist die Differenz zwischen der Deadline und der task request time des Tasks(also wenn der Task bereit zur Ausführung wird).

Das Adversary Argument besagt, dass es generell nicht möglich ist einen optimalen, komplett online basierten, dynamischen Scheduler zu konstruieren wenn mutual exclusion Bedingungen zwischen periodschen und sporadischen Tasks existieren und man die zukünftigen request times nicht kennt. Einen scheduler der alle zukünftigen request times kennt nennt man clairvoyant (=allwissend).

11.3 Mutual exclusion bei deadline Kap 11 S230

Es ist laut dem adversary argument unmöglich einen optimalen online scheduler zu konstruieren wenn es mutual exclusion constraints zw. sporadischen und periodischen Tasks gibt. (Sporadische Tasks müssen warten bis der periodische Task fertig ausgeführt wird. Da der sporadische Task eine laxity von 0 hat wird er die deadline verpassen) laxity: deadline interval $d_i - c_i$. Der deadline interval ist die Differenz zwischen der Deadline des Tasks und der Zeit wann ein Task zur Ausführung bereit ist.

Oder meint die Frage das hier: Um Tasks die mutual exclution Bedingungen haben gibt es 3 Wege um damit mittels eines dynamischen Scheduler umzugehen:

- Providing extra resources such that simpler sufficient schedulability tests and algorithms can be applied
- Dividing the scheduling problem into two parts such that one part can be solved offline at compile time and only the second (simpler) part must be solved at run time
- Introducing restricing assumptions concerning the regularity of the task set.

11.4 Rate Monotonic Algorithm – Formel, Voraussetzungen Kap 11 S 231

Der Rate Monotonic Algorithm ist ein dynamischer, preemtive algorithm der auf statischen Prioritäten beruht und ist optimal für SP Systeme. Vorraussetzungen sind:

- The request times for all tasks of the task set T_i for which hard deadlines exist are periodic
- All tasks are independent of each other. There exists no precedence constrains or mutual exclusion constraints between any pair of tasks
- The deadline interval of every task T_i is equal to its period p_i
- The required maximum computation time of each task c_i is known *a priori* and is constant
- The time required for context switching can be ignored
- The sum of the utilization factors μ of the n tasks is given by:

$$\sum_{i=1}^{n} \frac{c_i}{p_i} <= n(2^{\frac{1}{n}} - 1)$$

c_i ist die WCET Die Priorität des Tasks basiert auf der Periode des Tasks.

11.5 Scheduling - forbidden regions? Kap 11 S 233

Wenn es einen gegenseitige Ausschluss zwischen 2 Tasks gibt und Tasks erst nach Ablauf einer gewissen Zeit unterbrochen werden können kommt es durch zu Konflikten wenn einer der Tasks sofort behandelt werden muss. Wenn der Scheduler die zukünftigen request times dieses Tasks kennt kann er vor dessen Ausführung eine "forbidden region" einfügen damit der andere Task erst nach dessen Ablauf gestartet werden kann.

11.6 Priority Ceiling (Wie funktioniert, hinreichende sched. test für p. c.? Wann Task in c.s.) Kap 11 S 234

Das priority ceiling protocol wird verwendet um periodische Tasks welche exklusiven Zugriff auf eine geteilte Resource, die durch Semaphoren geschützt werden, benötigen. Das *priorityceiling* einer Semaphore ist definiert als höchste Priorität des Tasks der diese Semaphore locken darf. Ein Task T darf den kritischen Abschnitt nur dann betreten wenn seine Priorität höher ist als das priority ceiling aller Semaphoren die derzeit von anderen Tasks geblockt sind. Wenn ein low priority Task der sich in der kritische Section befindet Tasks mit höherer Priorität blockt wird ihm dessen Priorität für die Dauer seiner critical section zugewiesen. Dadurch wird verhindert dass ein task mit mittlerer Priorität (der diese Ressource nicht benötigt) während ein Task mit hoher Priorität der diese Ressource benötigt ausführen darf und damit den task mit der höchsten Priorität weiter verzögert.

Der Schedulability Test für dieses Protokoll sieht folgendermaßen aus:

$$\forall i, 1 \leq i \leq n : (c_1/p_1 + c_2/p_2 + ... + c_i/p_i + B_i/p_i) \leq i(2^{1/i} - 1)$$

Preemption durch high priority tasks wird durch die ersten i Terme ausgedrückt und die worstcase blocking time durch B_i/p_i.

Das priority ceiling protocol ist ein berechenbares aber *nicht deterministisches* Scheduling Protocol.

11.7 Scheduling online (Wie zeigt man, dass kein optimaler on-line scheduler existiert? Kap 11 S 230)

Das Adversary Argument besagt, dass es generell nicht möglich ist einen optimalen, komplett online basierten, dynamischen Scheduler zu konstruieren wenn mutual exclusion Bedingungen zwischen periodschen und sporadischen Tasks existieren.

"Beweis": Wenn ein periodischer Task abgearbeitet wird muss der sporadic task warten bis dieser fertig ist. Da ein sporadic task eine laxity von 0 besitzt verpasst er seine Deadline.

12 Validation

13 System Design

14 The Time-Triggered Architecture

A Fragenkatalog Ausarbeitung Theorie

A.1 Nachricht stabilität

??????????????????????????? vielleicht nur in den folien?

Token hält abstürzt. Dann ist die Kommunikation unterbrochen bis ein Node den Verlust erkennt und einen neuen Token generiert.

A.2 Transaktion (RT vs. DB)

A.3 Vektorzeit/ Lamportzeit (ZusammenhangKausalität, temp. Ordnung, V.- und L. zeit?)

???? nur folien?

A.4 Was ist ein SOS Fehler

A.5 sonstiges, vielleicht nur auf folien

A.6 Waist Line Architecture

A.7 Welche Interfaces hat eine Komponente (LIF/RS, DM, CP, lokale Interfaces). Was muss im LIF spezifiziert werden (Operation, Meta, 4-Universe Modell

B Fragenkatalog Ausarbeitung Praxis

B.1 Adversary Argument

Ein necessary schedulability test:

$$\mu = \sum \frac{c_i}{p_i} \leq n$$

μ_i ist der Auslastungsfaktor von Task T_i. c_i ist die execution time p_i ist die periode n Anzahl der Prozessoren

B.2 CAN

B.2.1 CAN max Leitungslänge

gegeben Bitrate (50kbit/s)
gesucht: maximale Leitungslänge

$bit\_length = \frac{Lichtgeschwindigkeit}{Datenrate} = \frac{200*10^6}{50*10^3} = 4000m$
Da 2x das Propagation Delay gewartet werden muss bis ein Bit stabil anliegt und sonstigen Overhead nimmt man einfach $1/3 - 1/4$ davon um die maximale praktische Länge zu bestimmen

B.2.2 CAN max Bandbreite

gegeben maximale Leitungslänge 80m
gesucht:Bitrate

$Datenrate = \frac{Lichtgeschwindigkeit}{Leitungslaenge} = \frac{200*10^6}{80} = 2,5Mbit/s$
Da 2x das Propagation Delay gewartet werden muss bis ein Bit stabil anliegt und sonstigen Overhead nimmt man einfach $1/3-1/4$ davon um die maximale praktische Datenrate zu bestimmen.

B.3 FTA Algorithmus

Angabe: Given a reading error of 20ms(20*10$\hat{}$(-3)s) and a resonator quality of 0,00001 and a resynchronisation period of 1 second. What precision can be achieved in a system with 10 clocks considering tha 1 clock is malicious.

Theorie Ablauf: Ein System mit N Nodes wo k Byzantinische Faults toleriert werden sollen. (Es werden $N \geq (3k+1)$ Nodes benötigt) Alle Nodes schicken ihre Timestamp an alle anderen. Eine Konvergenzfunktion wird ausgeführt die den Fehler der Inkonsistenz aller Uhren eines Systems minimiert und k Byzantinische Fehler toleriert: Jeder Node misst dabei den Zeitunterschied zwischen seiner Uhr und den Uhrzeiten aller anderen Nodes. Die k größten und kleinsten Unterschiede werden entfernt. Von den restlichen $N - 2k$ Unterschieden wird der Durchschnitt gebildet der dann der Korrekturwert für den Node ist.
Werte/Formel:
Table for typical Jitter ε:

synchronization message assembled and interpreted	approximate range of jitter
at the application software level	$500\mu sec$ to 5 $msec$
in the kernel of the operating system	10 μsec to 100 μsec
in the hardware of the communication controller	less than $10\mu sec$

Das impossibility result gibt an dass es nicht möglich ist Uhren intern auf eine bessere Präzision als

$$\Pi = \varepsilon(1 - \frac{1}{N})$$

zu synchronisieren.

Die Präzision des FTA berechnet sich: Eine Uhr mit einem byzantinischen Fehler verursacht den Fehler:

$$E_{byz} = \Pi/(N - 2k)$$

Eine Uhr mit k byzantinischen Fehler verursacht den Fehler:

$$E_{kbyz} = k\Pi/(N - 2k)$$

Die Konvergenzfunktion ist dann gegeben durch:

$$\Phi(n, k, \varepsilon) = (k\Pi/(N - 2k)) + \varepsilon$$

Die Präzision von FTA ist dann gegeben durch:

$$\Pi(N, k, \varepsilon, \Gamma) = (\varepsilon + \Gamma)\frac{N - 2k}{N - 3k} = (\varepsilon + \Gamma)\mu(N, k)$$

B.4 Limit to time measurement

- Wenn ein event von 2 unterscheidlichen Nodes beobachtet wird dann dann die Timestamp mit einem tick unterschiedlich sein. bei nur einem Tick unterschied kann die zeitliche Ordnung von der Timestamp nicht abgeleitet werden
- Wenn die beobachtete Dauer eines intervals d_{obs} ist dann ist die wahre duration d_{true} begrenzt durch:

$$(d_{obs} - 2g) < d_{true} < (d_{obs} + 2g)$$

- Die zeitliche Ordnung von events kann aus ihren Timestamps wiederhergestellt werden wenn die Differenz zwischen den Timestamps größer als 2 Ticks ist.
- Die zeitliche Ordnung kann immer wiederhergestellt werden wenn das set mindestens 0/3g precedent ist.

B.5 Priority Ceiling mit 2 Tasks und zwei Semaphoren

Der Schedulability Test für dieses Protokoll sieht folgendermaßen aus:

$$\forall i, 1 \leq i \leq n : (c_1/p_1 + c_2/p_2 + ... + c_i/p_i + B_i/p_i) \leq i(2^{1/i} - 1)$$

Preemption durch high priority tasks wird durch die ersten i terms ausgedrückt und die worstcase blocking time durch B_i/p_i.

B.6 Umwandeln Sporadic Task in Quasi periodic Task

Es ist möglich eine Lösung zu dem scheduling Problem zu finden wenn unabhängige sporadische tasks laxity l haben.

Parameter	sporadic task	new pseudo-periodic task
Computation time c	c	$c' = c$
Deadline interval d	d	$d' = d$
Period p	p	$p' = Min(l - 1, p)$

B.7 Minimal Latency Jitter

CAN Bus mit 5 Nodes: Dauer zum senden einer Nachricht: 0,1msec Minimale Dauer zwischen 2 Nachrichten von einem Node: 1msec Wie groß ist der jitter maximal.
$d_{min} = 0,1msec$ - Message hat höchste Priorität und is gleich dran
$d_{max} = 0,5msec$ - Alle anderen Nodes dürfen vorher schicken
$d_{jit} = d_{max} - d_{min} = 0,4msec$
Die minimale Dauer zwischen 2 Nachrichten ist uns egal, die ist nur dafür da dass kein Node 2 Nachrichten schickt im Berechnungszeitraum

B.8 Zeitmessung (Grundlegende 4 Schränken der Zeitmessung) Kap 3

Es ist ein System mit einer „reasonable“ globalen Zeitbasis mit Granularität g gegeben. Dann müssen die folgenden Grenzen der Zeitmessung beachtet werden:
· Wenn ein einzelnes Ereignis von zwei Knoten wahrgenommen wird, dann gibt es immer die Möglichkeit, dass sich die Zeitpunkte (timestamps) um einen Tick unterscheiden.

· Angenommen dobs ist die beobachtete (observed) Dauer eines Intervalls, dann ist die wahre Dauer (true duration): $(d_{obs}–2g) < d_{true} < (d_{obs} + 2g)$

· Die zeitliche Reihenfolge von Ereignissen kann nur dann wiederhergestellt werden, wenn die beobachtete Zeitdifferenz $d_{obs} >= 2g$ ist.

· Die zeitliche Reihenfolge von Ereignissen kann immer wiederhergestellt werden, wenn das Ereignispaar (event set) 0/3g präzident ist.

B.9 Hinweis zur nachfolgenden Praxiszusammenfassung!!!!!

Das Ergebnis von Beispiel 1.B ist falsch! Das Ergebnis ist $A_0\Pi$ und nicht $A_0\Pi * 10^{-1}$.Der Kollege hat sich da beim Umrechnen vertan
Dass es nicht stimmen kann kann man sich leicht überlegen da wir eine Tangente in den Wendepunkt der Funktion legen (dort hat die Tantente bei einer Sinus Funktion die höchste Steilheit) der genau bei $p/2$ liegt. Da wir mit 1/2 Periode abtasten ist der Bereich den wir betrachten genau zwischen den Extremewerten. Da sich die Funktion nahe bei den Extremwerten abflacht "überschießen" wir mit der Tangente und deshalb bekommen wir einen Wert der größer als die Amplitude ist. D.h die Funktion zur Näherung des Measurement Errors ist in dem Fall inadequat.

182.380 Echtzeitsysteme WS 2.0h

(Praxis)

1. Minimal Latency Jitter(S.9, 1.3.2; S.27, 1.7)

   A) Consider a RT entity that changes its value periodically according to $v(t)=A_0\sin((2\pi/T)*t)$ where T, the period of oscillation, is 100msec. What is the maximum change of the value of this RT entity within a time inverval of 1 msec?(express the result in percentage of the amplitude A_0)

   The "Additional Measurement Error" follows the equation

$$\Delta T = dT(t)/dt * \Delta d$$

Δd ... *Jitter*, $dT(t)/dt$... *Differential Quotient of the amplitue value*

   First we have to build v'(t)

$$v(t)=A_0\sin(2\pi f)t; (f=1/T)$$

$$v'(t)=A_0[2\pi f*\cos(2\pi f)t]$$

   We know that the maximum value during one period T of the cosinus function is $\cos(0\pm T)=1$.

$$v'(t)=A_0(2\pi f*1)$$

   Now it is easy to calculate ΔT

$$\Delta T = v'(t)*\Delta d$$
$$\Delta T = A_0(2\pi f)*10^{-3}; 1\text{msec}=1*10^{-3}\,sec$$
$$\rightarrow f=1/T=1/(100*10^{-3})$$
$$\rightarrow [1/(100*10^{-3})]*(10^{-3})=10^{-2}$$

$$\Delta T = A_0(2\pi*10^{-2}) \quad (\sim 6\%)$$

   The "Maximum Measurement Error" ΔT during one sampling period is about 6% of the

amplitude A_0 .

B) Consider a sine-function $v(t)=A_0\sin(2\pi f t)$ with the freuquency f = 100Hz and a sampling period $\Delta d=5\text{msec}$.
What is the "Maximum Measurement Error" ΔT of the RT entity during one period?(express the result in percentage of the amplitude A_0).

$$\Delta d=5\text{msec}=5*10^{-3}sec$$

$$v'(t)=A_0[2\pi f*\cos(2\pi f t)]=A_0*2\pi*100\,;(\rightarrow\cos(0\pm T)=1)$$

$$\Delta T=A_0(2\pi*100*0{,}5)*10^{-3}=A_0(\pi*10^{-1})\quad(\sim 30\%)$$

The "Maximum Measurement Error" ΔT during one sampling period is about 30% of the amplitude A_0 .

2. Protocol Efficiency

A) A given RT-Bus has the following properties:
Bus.bandwidth = 5Gbit/s ($5*10^9 bit$)
Bus.length = 10km ($Propagation\,Delay=50*10^{-6}$ s)
Message.length = 1000 bit
Determine the data efficiency of the bussystem.

.) $data\,efficiency<message.length/(message.length+bitlength\,of\,the\,channel)$

.) We know that it takes a signal about 5 μsec to travel across a channel of 1km length. (Time interval it takes a bit to travel from one end of the channel to the other end.)

.) $bitlength\,of\,a\,channel=bandwidth*propagation\,delay$
(The number of bits that can traverse the channel within one propagation delay)

With that knowledge, we now, can calculate the data efficiency:

a) Bitlength of the channel:

$$Bus.bitlength=5*50*(10^9*10^{-6})=250*10^3(250\text{kbit})$$

b) Data Efficiency

$$1/(1+250)=1/251\quad(0{,}4\%)$$

We see that the efficiency is very poor and only lies by about 0.4%. The reason for that is, because of the great amount of bandwidth together with relatively smallsized messages. So the usage of the bandwidth is very low and so the system is very inefficient

B) A given RT-Bus has the following properties:

Bus.bandwidth = 1Gbit/s ($1*10^9 bit$)

Bus.length = 1km ($Propagation Delay = 5*10^{-6}$ s)

Message.length = 1000 bit

Determine the data efficiency of the bussystem.

a) Bitlength of the Channel:

$$Bus.bitlength = 1*5*(10^9*10^{-6}) = 5*10^3 (5\text{kbit})$$

b) Data Efficiency:

$$1/(1+5) = 1/6$$

This System has only a data efficiency of ~ 17%.
(Kann man Efficiency von Netzwerk mit großen Kabellängen durch Erhöhen der Bandbreite(z.B Glasfibrewire) beliebig erhöhen?

Wie wir aus den Zusammenhängen aus (2.A) wissen ist dies nicht möglich, weil bei größerer Bandbreite bei gleich bleibender relativ kleiener Messags-Length die Auslastung stark zurückgeht und dadurch die Effizienz der Übertragung zurück geht.)

C) A given RT-Bus has the following properties:

Bus.bandwidth = 10Mbit/s ($10*10^6 bit$)

Bus.length = 1km ($Propagation Delay = 5*10^{-6}$ s)

Message.length = 100 bit

Determine the data efficiency of the bussystem.

a) Bitlength of the Channel:

$$Bus.bitlength = 10*5*(10^6*10^{-6}) = 50\text{bit}$$

b) Data Efficiency:

$$100/(100+50) = 2/3 \quad (66{,}6\%)$$

We now see that if we accommodate the messagelength and the bandwidth of the system the efficiency of the system grows.

3.) Real-Time Scheduling

A) Adversary Argument: “Es sagt generell aus, dass es nicht mÃ¶glich ist einen optimalen dynamischen Scheduler zu konzipieren, wenn es Mutual Exclusion Bedingungen zw. periodischen und sporadischen Tasks gibt.”

Periodischer Task: besitzt konstantes Zeitintervall zwischen aufeinanderfolgenden Task Resquests. Das bedeutet, dass nach dem Initial Request alle Task Request Times a priori bekannt sind.

Sporadischer Task: Es sind die Task Request Times nicht bekannt, jedoch weiß man zumindest, dass zwischen zwei aufeinanderfolgenden Requests ein minimales Zeitintervall liegt.

Aperioischer Task: Es liegen gar keine zukünftigen Informationen vor.

Optimaler Scheduler: Ein dynamischer online Scheduler ist optimal wenn er eine Lösung findet, wenn auch ein clairvoyant Scheduler, der totales Wissen über alle zukünftigen Task Request Times hat, eine Lösung finden würde.

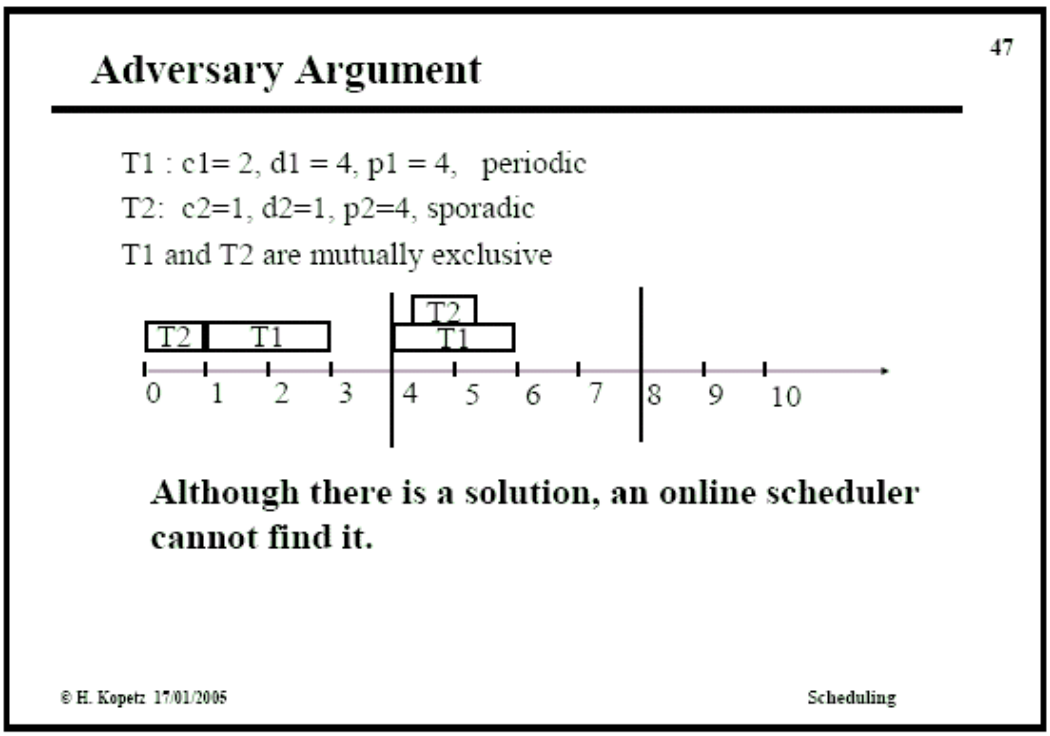
Notwendiger Schulability Test:

$$\mu = \Sigma\, c_i / p_i \leq n \quad n \ldots Anzahl\ der\ Prozessoren$$

$l_i \ldots d_i - l_i$ *nennt man die Laxity eines Tasksund sie beschreibt dieWCET*

$c_i \ldots$ *Execution Time ;* $p_i \ldots$ *Task Period ;* $d_i \ldots$ *Deadline Intervall* $(Deadline - TaskRequstTime)$

$\mu_i \ldots$ *Utilization Factor gibt an wieviel Prozent der Zeit Task Ti den Prozessor beansprucht*



$\mu = 2/4 + 1/4 = 3/4 \leq 1$ Obwohl die "Necessary Scheduling Condition" erfüllt ist findet der Online-Scheduler keine Lösung, obwohl eine existiert.

B) Dynamisches Scheduling

Der Rate Monotonic Algorithmus:
Folgende Annahmen müssen getroffen werden

1. Die Requests für alle Tasks aus dem Task Set { T_i } für die harte Deadlines existieren, müssen periodisch sein.
2. All Tasks sind unabhängig von einander. Es existiert keine bestimmte Reihenfolge oder Mutexbedingungen zwischen den Tasks.
3. Das Deadline Invervall jedes Tasks T_i ist gleich der Periode p_i .
4. Die benötigte Maximale Ausführungszeit c_i jedes Tasks ist *a prioir* bekannt und konstant
5. Die Zeit die für Context Switching beansprucht wird kann ignoriert werden.
6. Die Summe der Utilization Factors μ von n Tasks ist bestimmt durch

$$\mu = \Sigma c_i / p_i \leq n(2^{(1/2)} - 1)$$

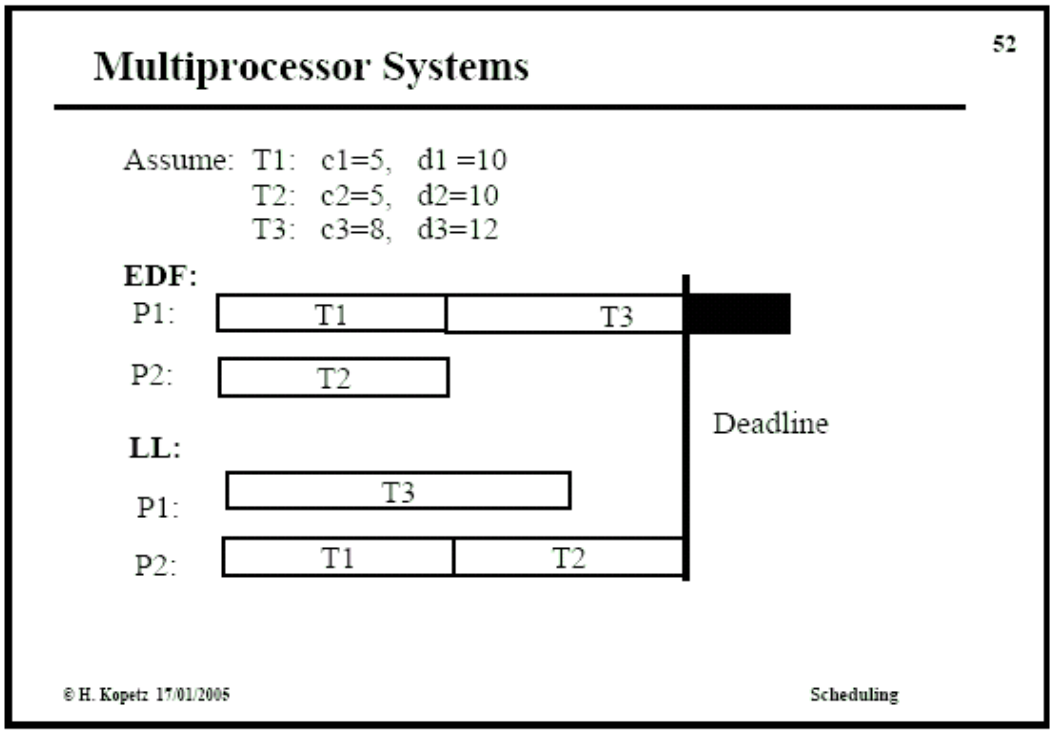
Der Term n(2^(1/2) – 1) nächert sich den Wert ln 2 an der etwa 0.7 beträgt für n gegen Unendl.

Diese Gruppe der Scheduling-Algorithmen entscheidet on-line welcher Task aus

dem Ready-Pool als nächster geschedult wird.

a) Earliest Deadline First(EDF)

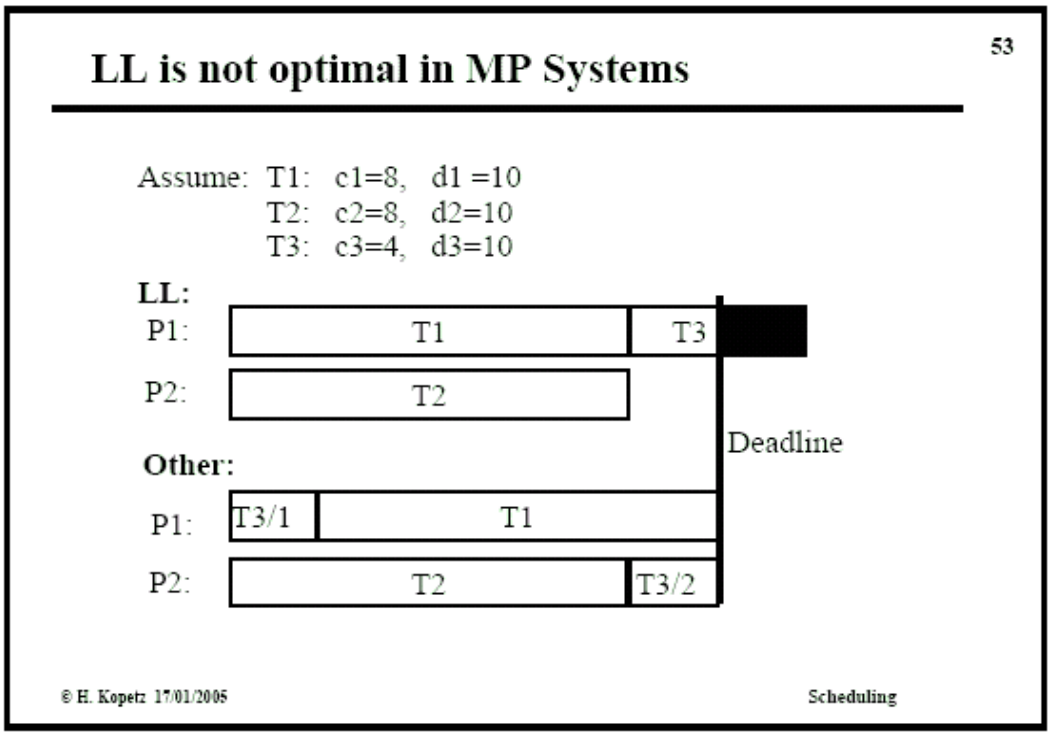
Annahmen 1 – 5 müssen gelten
ist ein dynamischer preemtiver Scheduling-Algorithmus mit dynamischen Prioritäten.
Der Task mit der frühesten Deadline bekommt die höchste dynamische Priorität
EDF ist in Uniprozessorsystemen optimal.
Bei Multiprozessorsystemen kann es zu folgenden Problem kommen:



b) Least Laxity(LL)

Ist auch ein dynamische preemtiver Scheduling-Algorithmus mit dynamischen Prioritäten und trifft die gleichen Annahmen wie der EDF Algorithmus und ist in Uniprozessorsystemen optimal hat jedoch auch Probleme mit Multiprozessorsystemen. Hier werden die Prioritäten durch die Laxity(siehe oben) bestimmt und der Task mit der kürzesten Laxity bekommt die höchste

dynamische Priorität.



4. Real-Time Commonication

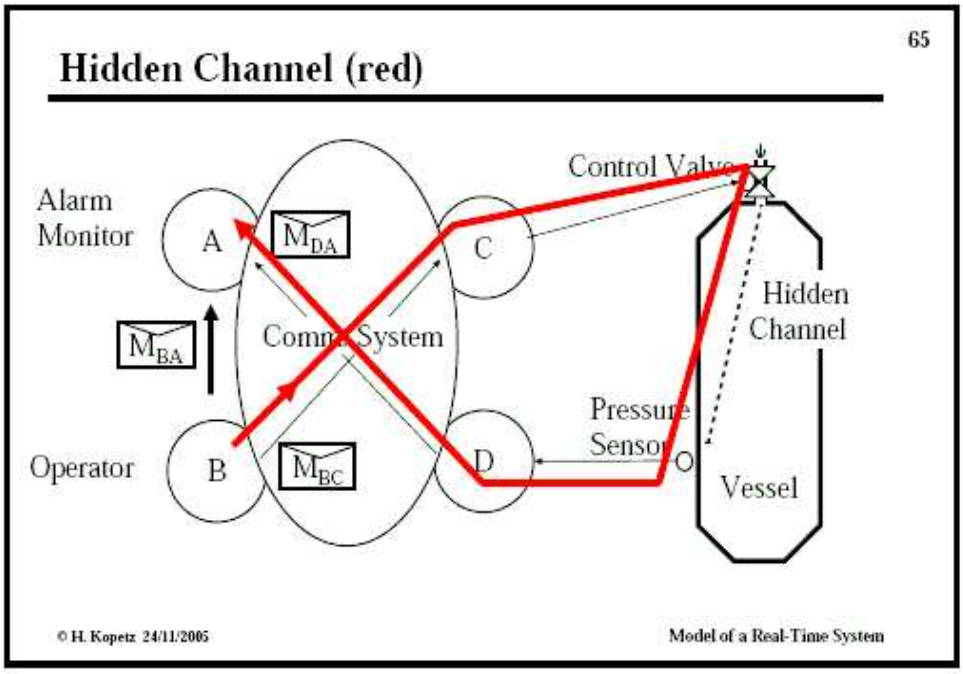
A) Action Delay of PAR:

Theoretische Grundlagen:

a) Permanence of a Message: Beschreibt Relation zwischen bestimmter Nachricht die an einen Knoten angekopmmen ist und dem Set von Nachrichten die zeitlich vor dieser bestimmten Nachricht an den Knoten gesendet wurden. Die Nachricht wird genau an dem Zeitpunkt permanent, wenn der Empfänger weiß dass, alle Nachrichten die voher gesendet wurden bereits eingetroffen sind oder nie ankommen werden.

b) Action Delay: Maximales Zeitintervall zwischen Sendezeitpunkt und permanent werden einer Nachricht.

c) Hidden Channel:(siehe S109)



$t_{permanent} = t_{send} + d_{max} + 2g$ *für System mit Globaler Zeit*

$t_{permanent} = t_{send} + d_{max} + (d_{max} - d_{min}) + g_l$ *für System ohne globaler Zeit*

In einem System ohne globaler Zeit hat der Receiver keine Information wann die Message gesendet wurde(mit globaler Zeit kann dieser Zeitpunkt Teil der Message sein) und muss daher auch noch zusätzlich den Protocol Jitter $\varepsilon(d_{max} - d_{min})$ abwarten.

Beispiel(siehe S151)

TRT = 10 msec; Retry Count = 2
Zeit um Message auf den Bus zu Transportieren = 1msec

Für eine komplette Übertragung einer Nachricht muss der bidirectionale Kanal 2x passiert werden und 2x eine Nachricht(Message u ACK) auf den Bus gelegt werden
--> Timeout Intervall des PAR: 2*10 msec + 2*1msec = 22msec

$d_{min} = 1\text{msec}$ (bei der Anfrage des Cient zum Senden einer Nachricht ist der Client gerade in Besitz des Token und man hat nur 1msec zum legen der Nachricht auf den Bus)

$d_{max} = 22\text{msec} + 22\text{msec} + TRT + 1\text{msec} = 55\text{msec}$ (Auf diesen Wert kommt man folgnder Weise: wegen der 2 Retries haben wir die 2x22msec die wir beim Time-Out warten; die längste Zeit die wir auf den Token warten müssen wäre 1xTRT=10msec; dann haben wir auch noch 1msec zum legen der Daten auf den Bus)

$d_{jit} = d_{max} - d_{min} = 55\text{msec} - 1\text{msec} = 54\text{msec}$

Mit diesen Werten können wir jetzt das Action Delay berechenen:

a) ohne globaler Zeit:

$t_{permanent} = t_{send} + d_{max} + d_{jit} + g_l = 55\text{msec} + 54\text{msec} + t_{send} + g_l = 109\text{msec} + t_{send} + g_l$

b) mit globaler Zeit($g = 100\,\mu sec = 0.1\,msec$):

$t_{permanent} = t_{send} + d_{max} + 2\text{g} = 55\text{msec} + 0.2\text{msec} + t_{send} = 55.2\text{msec} + t_{send}$

Error Detection Latency(Zeit bis ein Fehler entdeckt wird bei k=2):

$t_{err} = 3 * Timeout\ Intervall = 66\text{msec}$

Beispiel(Folien: RT-Communicaton/20):

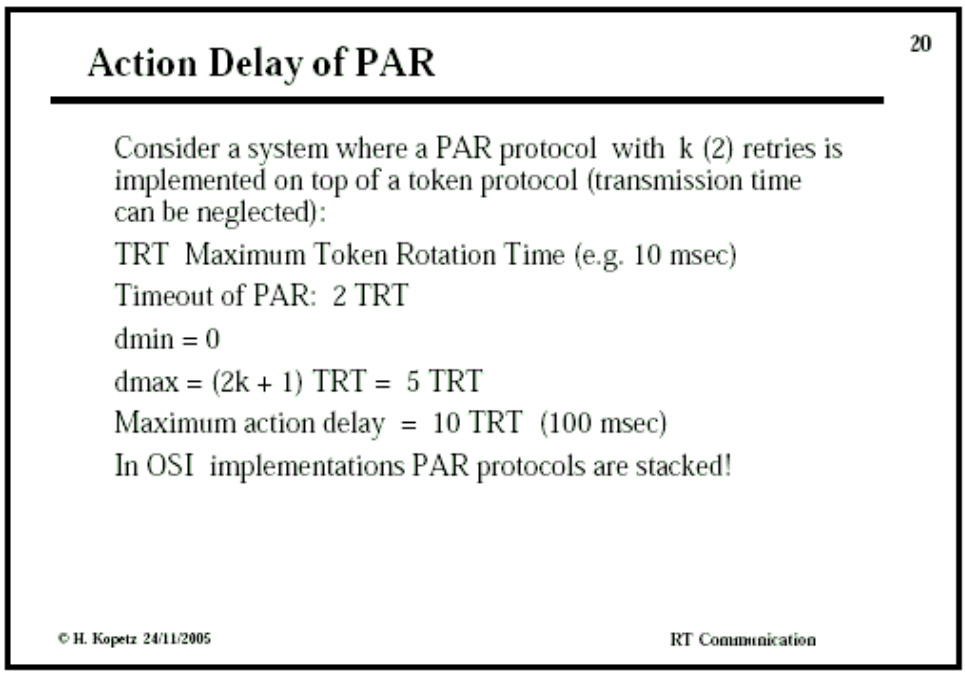
TRT = 10msec; Transmission Time = 0; no global time
Timeout Intervall = 10msec + 10msec = 20msec

$d_{min} = 0\text{msec}$

$d_{max} = 2 * Timeout\ Invervall + TRT + 0 = 40\text{msec} + 10\text{msec} = 50\text{msec}$

$d_{jit} = d_{max} - d_{min} = 50\text{msec}$

$t_{permanent} = d_{max} + d_{jit} + t_{send} + 2\text{g} = 50\text{msec} + 50\text{msec} + t_{send} + 2\text{g} = 100\text{msec} + t_{send} + 2\text{g}$



Codierungsverfahren(Physical Layer)

.) NRZ-Code: ist ein Einfaches Codierungsverfahren bei dem log. "0" und log. "1" einfach durch die Signalpegel "high" und "low" kodiert werden.
für Syschronisierung ist sog. "Bitstuffing" notwendig was aber für die DataEfficiency eher schlecht ist, da dadurch die Länge einer Message sehr datenabhängig wird
-) non-synchronizing: Falls Code nur 0en oder 1en enthält, finden am Kanal keine Transitionen statt und der Empfänger kann nicht die Clock Ticks des Empfängers erkennen.
+) einfach zu realisieren
+) FeatureSize: 1 Bitcell

.) Manchaster Code: es befindet sich innerhalb einer Bitcell immer eine sog. Synchronisationedge die es ermöglich dem Empfänger die Glockticks des Sender zu erkennen; log "1" wird als low/high Edge, log "0" high/low Edge
+) synchronizing
-) FeatureSize:1/2 Bitcell

.) Modified Frequency Modulation(MFM):
es wird zwischen ClockPoints und DataPoints unterschieden
log. "0" wird als keine Pegeländerung an einem DataPoint kodiert und log. "1" wird als Pegeländerung an einem DataPoint kodiert. Falls mehr als 2 "0" in Folge

auftreten verlanget es das Kodierungschema das an einem Clockpoint eine Pegeländerung auftritt.
+) synchronizing
+) FeatureSize: 1 Bitcell
-) kompliziert und aufwendig zu realisieren

(siehe Buch/chapter7/S167-168)

Warum wollen wir eigetl. keine kleine FeatureSize?
Dies ist damit verbunden dass der physik Aufbau des Codes auch die EMI stark beeinflusst.
Kleine FeatureSize erfordert steile Flanken und muss dadurch durch Höherfrequente Signale dargestellt werden was die EMI erhöht, was auch zur Folge hat dass solche Codes nicht bei Hochfrequenten Übertragungen eingsetzt werden können.

4. Dependability Requirenments

Def.: Das Vertrauen das ein User in die zur Verfügung gestellten Dienste des Computersystems setzen kann.

a) Reliability:

Def.: Die Wahrscheinlichkeit das ein Computersystem seinen Service bis zum Zeitpunkt t noch korrekt liefert, vorausgesetzt es war zum Zeitpunkt $t=t_0$ funktionsfähig

Falls das System eine konstante Fehlerrate λ *failures*/*hour* dann ist die Reliability R(t) zum Zeitpunkt t gegeben durch

$$R(t)=e^{(-\lambda(t-t_0))}$$

Das Inverse der Fehlerrate

$$1/\lambda=MTTF$$

wird als Mean Time to Failure bezeichnet und beschreibt die mittlere Zeitspanne in Stunden bis zum Auftreten eines Failures.

b) Safety:

Def.: Bezeichnet die Reliability in hinsicht auf kritische Fehler.
Kritische Failures werden als *malign* bezeichnet
Unkritische Failures werden als *benign* bezeichnet.
Beim Auftreten eins malign Failure Modes kann es oft der Fall sein, dass die Kosten die der Fehler verursacht, den Nutzen des Systems um mehrere Dimensionen übersteigt.

c)Maintainability

Def.: Ist ein Mass dafür wie lange es dauert ein System nach dem Auftritt eines benign Failures wieder in Stande zu setzten.
Sie wird wieder mit einem Wahrscheinlichkeitswert angegeben der mit M(d) bezeichntet wird. Es beschreibt die Wahrscheinlichkeit, dass ein System nach dem Auftritt eines Failures im Zeitintervall d wieder in Betrieb ist
Es existiert analog zur Fehlerrate auch eine Reperaturrate die mit dem Zeichen μ bezeichnet wird und eine Mean Time to Repair die der Kehrwert der Reparaturrate ist:

$$1/\mu = MTTR$$

Es gibt immer einen Konflikt zwischen Maintainability und Relatively, weil unaustauschbare Komponenten zwar eine größere Reliability aber dafür nicht wartbar sind.

d) Availability

Def.: Ist das Maß dafür wie Lange ein System einen korrekten Servive liefert in Hinblick auf die Abwechselnden Zeitmaße eines korrekten und inkorrekten Service.

$$A = MTTF/(MTTF + MTTR) = MTTF/MTBF$$

e) Security

Def.: Die Fähigkeit eines Systems in Hinblick auf Datenschutz und das Verhinderen von unauthorisiertem Zugriff auf das System.

X) Appendix

Übertragungsraten eines CAN Bus zum vergleich der errechneten Ergebnisse:

10kbit/s	6.7km
20kbit/s	3,3km
50kbit/s	1,3km
125kbit/s	530m
250kbit/s	270m
500kbit/s	130m
1Mbit/s	40m

Die maximale (theoretische) Leitungslänge beträgt z. B. bei 1 MBit/s 40 m, bei 500 kBit/s sind 100 m möglich und bei 125 kBit/s 500 m. Diese Maximalwerte beruhen darauf, dass die Zeit, die ein Signal am Bus anliegt (Bitzeit, Bit/Sekunde), umso kürzer ist, je höher die Übertragungsrate ist. Mit zunehmender Leitungslänge steigt jedoch die Zeit, die ein Signal braucht, bis es am anderen Ende

des Busses angekommen ist (Ausbreitungsgeschwindigkeit). Daher darf die Zeit, die ein Signal am Bus liegt, nicht kürzer sein als die Zeit, die ein Signal braucht um sich auszubreiten. Zu beachten ist, dass sich das Signal nicht nur ausbreitet, sondern auch innerhalb der Signalzeit der Empfänger auf den Sender reagieren muss (siehe ACK).
Der Sender muss wiederum die eventuelle Buspegeländerung des/der Empfänger mitbekommen (siehe auch Arbitrierung). Deshalb ist die max. Leitungslänge etwas komplexer zu berechnen. Es müssen Verzögerungszeiten auf der Leitung, des Tranceivers (Sender und Empfänger), des Controllers (Sender und Empfänger) und der gesetzte Abtastzeitpunkt (Sender und Empfänger) berücksichtigt werden

Umrechnugstabelle

milli	10^{-3}
mikro	10^{-6}
nano	10^{-9}
femto	10^{-12}
atto	10^{-15}
kilo	10^{+3}
mega	10^{+6}
giga	10^{+9}
tera	10^{+12}
peta	10^{+12}