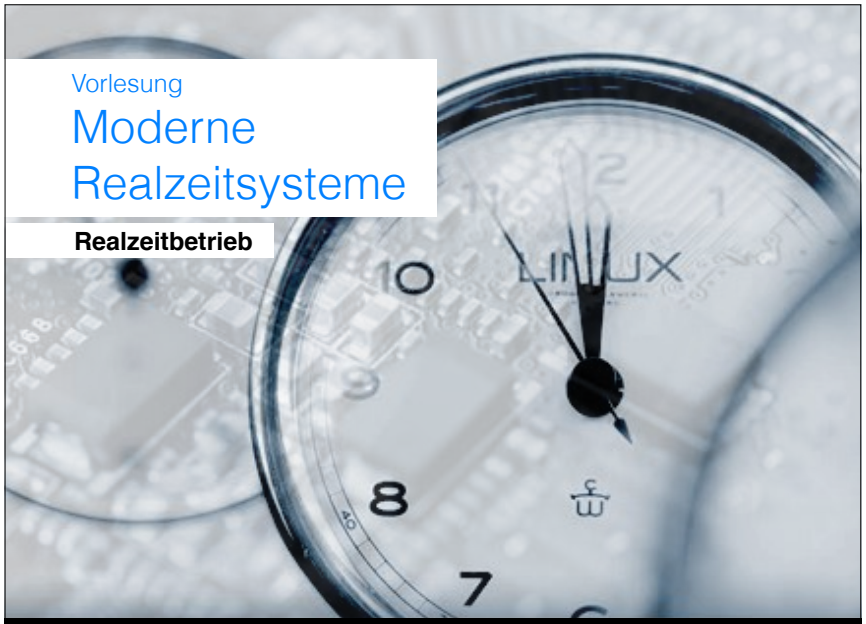


1

Lernziele

- Unterschiedliche Zeitbegriffe kennenlernen.
- Zeitliche Charakterisierung von
  - Außenwelt (technische Prozesse, Anforderungen),
  - Tasksets (Rechenprozesse, Lösungen) und
  - ausgewähltes Rechensystems.
- Aufstellen der beiden Realzeitbedingungen.
- Harte von weichen Realzeitsystemen unterscheiden können.
- Voraussetzungen, die einen Realzeitbetrieb ermöglichen, kennen.
- Einsatz und Umgang mit Ressourcen (gemeinsame Betriebsmittel).

2

Inhalt

- **Zentrale Beschreibungsgrößen**
  - Technischer Prozess
    - „Außenwelt“
  - Rechenprozess
    - „Taskset“
  - Systemsoftware
- **Realzeitbedingungen**
  - Auslastungsbedingung
  - Rechtzeitigkeitsbedingung
  - Harte- und weiche Realzeit
- **Systemaspekte**
  - Unterbrechbarkeit
  - Prioritäten
  - Ressourcen

3

Definition

Als **Realzeitsystem** wird die zentrale informations-verarbeitende Komponente (Reihe von Rechenprozessen, Taskgebilde, Steuerung) eines technischen Systems bezeichnet, welches neben den funktionalen Anforderungen auch **zeitlichen** Anforderungen genügen muss.

4

Zentrale Beschreibungsgrößen

1. **Beschreibungsgrößen des technischen Prozesses**
2. **Beschreibungsgrößen des entworfenen Taskgebildes (der Rechenprozesse)**
3. **Beschreibungsgrößen der Systemsoftware**

Zentrale Beschreibungsgrößen

1. **Beschreibungsgrößen des technischen Prozesses**
2. **Beschreibungsgrößen des entworfenen Taskgebildes (der Rechenprozesse)**
3. **Beschreibungsgrößen der Systemsoftware**

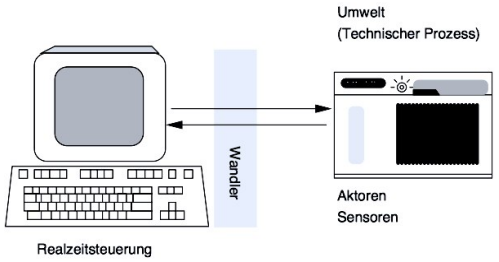
Zentrale Beschreibungsgrößen

Die **informationsverarbeitende Komponente** steht in Beziehung zur **Außenwelt** (dem technischen Prozess).

- Sie nimmt die Außenwelt über **Sensoren** wahr.
- Sie beeinflusst die Außenwelt über **Aktoren**.

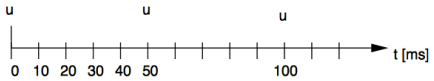
Zentrale Beschreibungsgrößen

- **Zwei Komponenten sind zu betrachten**
- Der technische Prozess (Außenwelt) stellt die **zeitlichen Vorgaben**.
- Die Realisierung des Taskgebildes auf einer spezifischen Hardware reflektiert die **zeitlichen Möglichkeiten**.



Zentrale Beschreibungsgrößen

- **Rechenzeitanforderung (technischer Prozess)**
  - Der technische Prozess löst Ereignisse aus (zum Beispiel „Druck zu hoch“), die von der Steuerung verarbeitet werden sollen. Dafür wird in der Steuerung Rechenzeit benötigt.
  - Rechenzeitanforderungen (Ereignisse des technischen Prozesses) werden durch Buchstaben gekennzeichnet.
  - In Grafiken wird der Zeitpunkt, an dem eine Rechenzeitanforderung auftritt, in der Regel durch den entsprechenden Buchstaben gekennzeichnet.



Zentrale Beschreibungsgrößen

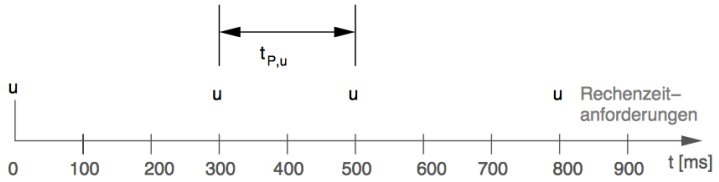
- Beispiel: Rechenzeitanforderung an einen modernen Fahrradcomputer
  - Ein Magnet in der Speiche löst jede Umdrehung eine Rechenzeitanforderung aus.
    - Kennzeichnung der Rechenzeitanforderung mit „u“.
  - Eingaben über das Touchscreen lösen Rechenzeitanforderungen aus.
    - Kennzeichnung der Rechenzeitanforderung mit „s“.

Zentrale Beschreibungsgrößen

- **Prozesszeit $t_{P,i}$**
  - Zeitliche Abstand, in dem zwei Rechenzeitanforderungen gleichen Typs auftreten.
  - Jede Rechenzeitanforderung „i“ hat eine Prozesszeit $t_{P,i}$.
  - Prozesszeiten sind selten konstant:
    - Minimale Prozesszeit: $t_{Pmin,i}$
    - Maximale Prozesszeit: $t_{Pmax,i}$
  - Die maximale Prozesszeit ist oft unendlich, belastet den Steuerungsrechner nicht und ist daher „uninteressant“.
  - Der Kehrwert der Prozesszeit ist die **Rate**

Zentrale Beschreibungsgrößen

- **Prozesszeit**

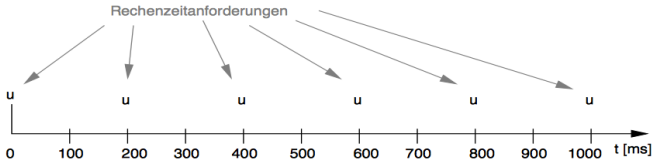


Zentrale Beschreibungsgrößen

- **Beispiel: Bestimmung der Prozesszeit**
  - Geschwindigkeitsanzeige Fahrradcomputer
  - Umfang der Räder eines Fahrrades: 1200 mm bis 2300 mm.
  - Maximal Geschwindigkeit 150 km/h
  - $t_{Pmax,u}$ =unendlich (Fahrrad wird nicht bewegt)
  - $t_{Pmin,u}$ = ?

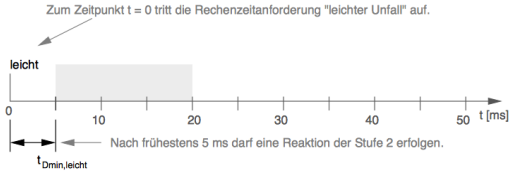
Zentrale Beschreibungsgrößen

- **Releasetime $t_{Release,i}$**
  - Auftrittszeitpunkt einer Rechenzeitanforderung.
  - Beispiel: Eine periodische Rechenzeitanforderung u mit einer Prozesszeit von $t_{Pmin,u}$ = 200 ms tritt zu den Zeitpunkten 0 ms, 200 ms, 400 ms, 600 ms usw. auf.



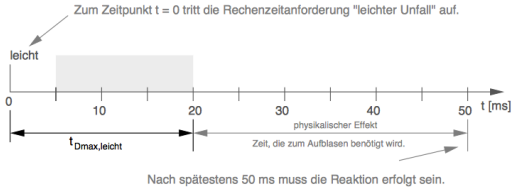
Zentrale Beschreibungsgrößen

- **Zulässige Reaktionszeit = Deadline**
  - Technische Prozess legt fest, ab welchem Zeitpunkt auf eine Rechenzeitanforderung reagiert werden darf (**minimale Deadline, $t_{Dmin,i}$**).
  - **Beispiel zweistufiger Airbag**: Bei einem modernen, zweistufigen Airbag darf – abhängig von der Unfallschwere – die zweite Stufe frühestens 5 ms nach Detektion des Unfalls erfolgen.



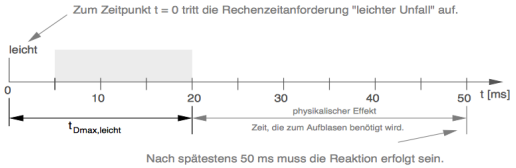
Zentrale Beschreibungsgrößen

- **Zulässige Reaktionszeit = Deadline**
  - Technische Prozess legt fest, bis zu welchem Zeitpunkt die Reaktion auf eine Rechenzeitanforderung erfolgt sein muss (**maximale Deadline, $t_{Dmax,i}$**).
  - **Beispiel Airbag**: Nach spätestens 20 ms muss der Airbag gezündet worden sein, sonst hat er keine verletzungsmindernde Wirkung.



Zentrale Beschreibungsgrößen

- **Zulässige Reaktionszeit = Deadline**
  - Physikalische Effekte sind heraus zu rechnen: Maximal zulässige Reaktionszeit ist der Zeitpunkt, zu dem der Rechner seine Werte ausgegeben hat
  - **Beispiel Airbag**: Zeit zwischen Unfall und Aufschlagen auf dem Lenkrad: 50 ms. Aufblasen des Airbags: 30 ms. Maximale Reaktionszeit $t_{Dmax,leicht}$ = 20 ms.



Zentrale Beschreibungsgrößen

- **Zulässige Reaktionszeit = Deadline (Fortsetzung)**
  - Oft: Maximale Reaktionszeit ist durch die Anforderung definiert, dass ein Ereignis vor dem Eintreffen eins nachfolgenden Ereignisses gleichen Typs bearbeitet sein muss.
  - In diesem Fall entspricht die maximal zulässige Reaktionszeit der minimalen Prozesszeit: $t_{Dmax,i} = t_{Pmin,i}$

Zentrale Beschreibungsgrößen

- **Phase**
  - Minimaler zeitlicher Abstand zwischen einer Rechenzeitanforderung i zum Zeitpunkt 0.
  - Ist die Phase = 0: Beide Ereignisse sind voneinander unabhängig (default).
  - Beispiel Fahrradcomputer:
    - Rechenzeitanforderungen u (Umdrehung des Vorderrades) und s (Eingabe Touchscreen) sind voneinander unabhängig, die Phase $t_{Ph,s}$ = 0.

Zentrale Beschreibungsgrößen

- **Zusammenfassung Beschreibungsgrößen „Außenwelt“**
  - Rechenzeitanforderung i
  - Prozesszeit $t_{P,i}$, insbesondere $t_{Pmin,i}$ (auch Periode genannt)
  - Releasetime $t_{Release,i}$
  - Minimal zulässige Reaktionszeit $t_{Dmin,i}$
  - Maximal zulässige Reaktionszeit $t_{Dmax,i}$
  - Phase t_{Ph}

Zentrale Beschreibungsgrößen

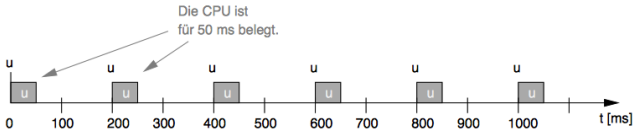
1. Beschreibungsgrößen des technischen Prozesses
2. **Beschreibungsgrößen des entworfenen Taskgebildes (der Rechenprozesse)**
3. Beschreibungsgrößen der Systemsoftware

Zentrale Beschreibungsgrößen

- **Ausführungszeit, Verarbeitungszeit, Executiontime t_E**
  - Summe der CPU-Zyklen, die zur Ausführung benötigt werden.
  - Bearbeitet ein CPU nur eine Task i, dann ist die Verarbeitungszeit $t_{E,i}$ die Differenz zwischen End- und Startzeit.
  - Die Verarbeitungszeit ist selten konstant. Sie schwankt zwischen einem minimalen Wert $t_{Emin,i}$ (**Best Case Execution Time, BCET**) und einem maximalen Wert $t_{Emax,i}$ (**Worst Case Execution Time, WCET**). Gründe für die Schwankungen sind:
    - Implementierung der verwendeten Algorithmen,
    - Caches (Daten- und Instruktionscaches, TLB, Pagecache, ...)

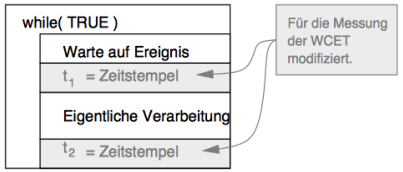
Zentrale Beschreibungsgrößen

- **Ausführungszeit, Verarbeitungszeit, Executiontime t_E**
  - Wird die Verarbeitungszeit t_E über der Zeit aufgetragen, erhält man die so genannte **Rechnerkernbelegung**.



Zentrale Beschreibungsgrößen

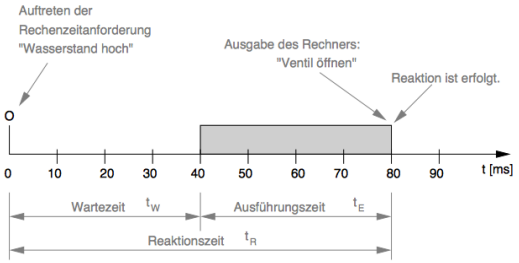
- **Ausführungszeit, Verarbeitungszeit, Executiontime t_E**
  - Die **Bestimmung** der BCET und der WCET ist eine der (schwierigen) Aufgaben des Informatikers bzw. Ingenieurs.
  - Statische Analyse von Quellcode und eingesetzter Hardware (in der Praxis kaum nutzbar).
  - Ausmessen der BCET beziehungsweise WCET.



Zentrale Beschreibungsgrößen

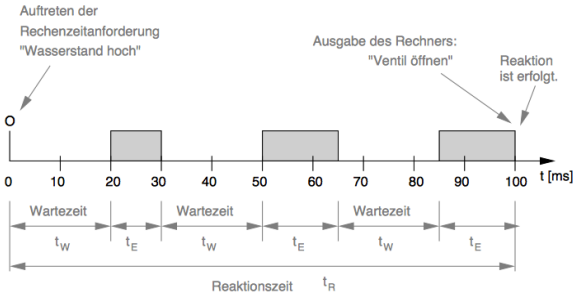
- **Reaktionszeit**
  - Zeit zwischen dem **Auftreten** einer Rechenzeitanforderung und dem **Ende** der Bearbeitung.

ACHTUNG: Nicht mit der Latenzzeit verwechseln!



Zentrale Beschreibungsgrößen

- **Reaktionszeit**
  - Die Reaktionszeit ergibt sich unabhängig davon, ob Tasks stückchenweise oder eben „am Stück“ abgearbeitet werden.



Zentrale Beschreibungsgrößen

- **Reaktionszeit**
  - Ja nach Auslastung des Systems ergibt sich eine minimale Reaktionszeit $t_{Rmin,i}$ und eine maximale Reaktionszeit $t_{Rmax,i}$.

Zentrale Beschreibungsgrößen

- **Zusammenfassung Beschreibungsgrößen „Taskset“**
  - Minimale Executiontime $t_{Emin,i}$ (BCET)
  - Maximale Executiontime $t_{Emax,i}$ (WCET)
  - Minimal zulässige Reaktionszeit $t_{Rmin,i}$
  - Maximal zulässige Reaktionszeit $t_{Rmax,i}$

Zentrale Beschreibungsgrößen

1. Beschreibungsgrößen des technischen Prozesses
2. Beschreibungsgrößen des entworfenen Taskgebildes (der Rechenprozesse)
3. **Beschreibungsgrößen der Systemsoftware**

Zentrale Beschreibungsgrößen

- **Latenzzeit**
  - Die Systemsoftware ist maßgeblich für das Zeitverhalten des Realzeitsystems mitverantwortlich.
  - Wesentliche Größe: Latenzzeit.
  - Zeit zwischen dem **Auftreten** einer Rechenzeitanforderung und dem **Start** der zugehörigen Bearbeitungsfunktion.

ACHTUNG: Latenzzeit wird häufig mit Reaktionszeit verwechselt.

Zentrale Beschreibungsgrößen

- **Folgende Latenzzeiten werden unterschieden**
  - Interrupt-Latenz t_{LIsr}
  - Task-Latenz t_{LTask}
  - Kernel-Latenz $t_{LKernel}$
  - Preemption-Delay

Realzeitbedingungen

1. **Gleichzeitigkeit und Auslastung**
2. **Rechtzeitigkeit (timeliness)**
3. **Harte und weiche Realzeit**

Realzeitbedingungen

1. **Gleichzeitigkeit und Auslastung**
2. **Rechtzeitigkeit (timeliness)**
3. **Harte und weiche Realzeit**

Realzeitbedingungen

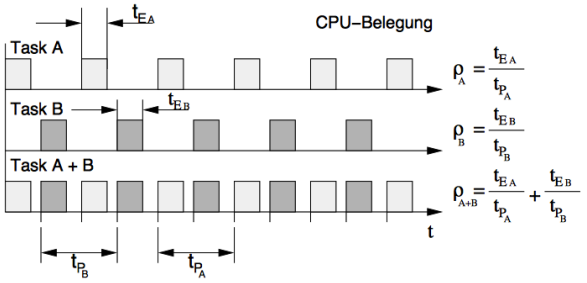
- **Ob eine Aufgabe in Realzeit, also schritthaltend, verarbeitet wird, lässt sich anhand zweier Bedingungen ableiten, nämlich**
  - der Auslastungsbedingung und
  - der Rechtzeitigkeitsbedingung.

Realzeitbedingungen

- Die Codesequenz (häufig auch mit Task gleichgesetzt), die eine Rechenzeitanforderung i bearbeitet, benötigt auf einer ausgewählten Plattform die Rechenzeit $t_{E,i}$.
- Diese Rechenzeit fällt mit der Prozesszeit der Rechenzeitanforderung $t_{P,i}$ an (jedesmal, wenn das zugehörige Ereignis auftritt).
- Der Quotient aus $t_{E,i}$ und $t_{P,i}$ entspricht der Auslastung durch die Rechenzeitanforderung auf der ausgewählten Plattform.

Realzeitbedingungen

- **Beispiel**
  - $t_{Emax,A}$ = 0.8 ms ; $t_{Pmin,A}$ = 2 ms ;
  - $t_{Emax,B}$ = 0.8 ms ; $t_{Pmin,A}$ = 2 ms ;
  - Gesamtauslastung 80 Prozent



Realzeitbedingungen

- Jede CPU der ausgewählten Rechnerplattform kann nicht mit mehr als 100 Prozent ausgelastet werden.
- Daher werden die Auslastungen sämtlicher Rechenzeitanforderung aufsummiert. Die Summe darf nicht größer als die Zahl der CPU-Kerne c beziehungsweise c*100 Prozent sein. Für eine Einkernmaschine ist c=1, für ein Hexacore c=6 (bzw. 600 Prozent).

Realzeitbedingung – Auslastungsbedingung

$$\rho_{ges,max} = \sum_{j=1}^{n} \frac{t_{Emax,i}}{t_{Pmin,i}} \leq c$$

Realzeitbedingungen

- **Auslastungsbedingung**
  - Die Auslastungsbedingung ist eine notwendige, aber keine hinreichende Bedingung. Sie ist Grundvoraussetzung für die schritthaltende Verarbeitung.

Realzeitbedingungen

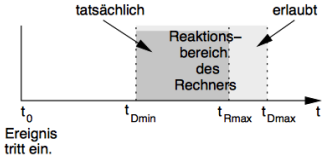
1. Gleichzeitigkeit und Auslastung
2. **Rechtzeitigkeit (timeliness)**
3. Harte und weiche Realzeit

Realzeitbedingungen

- **Rechtzeitigkeit**
  - Die Antwort auf eine Rechenzeitanforderung darf nicht zu früh und auch nicht zu spät erfolgen; sie muss pünktlich beziehungsweise rechtzeitig sein.
  - Nicht mit Schnelligkeit verwechseln! Schnelligkeit ist eine Hilfe dabei, rechtzeitig zu reagieren, sie ist nicht das Ziel.
  - Die „Außenwelt“ gibt mit $t_{Dmin,i}$ und $t_{Dmax,i}$ die zeitlichen Randbedingungen (Zeitfenster) für eine Rechenzeitanforderung i vor.
  - Innerhalb dieses Zeitfensters muss das von uns gebaute System reagieren.

Realzeitbedingungen

- **Rechtzeitigkeit**
  - Für den Realzeitbetrieb (schritthaltende Verarbeitung) muss die Reaktionszeit auf eine Rechenzeitanforderung im zugehörigen Zeitfenster liegen.



Realzeitbedingung – Rechtzeitigkeitsbedingung

Für jede Rechenzeitanforderung i muss gelten:

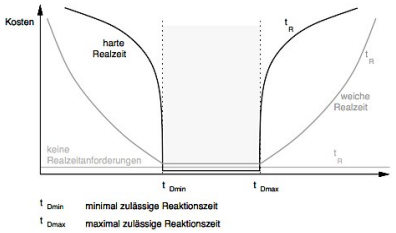
$$t_{Dmin,i} \leq t_{Rmin,i} \leq t_{Rmax,i} \leq t_{Dmax,i}$$

Realzeitbedingungen

1. Gleichzeitigkeit und Auslastung
2. Rechtzeitigkeit (timeliness)
3. **Harte und weiche Realzeit**

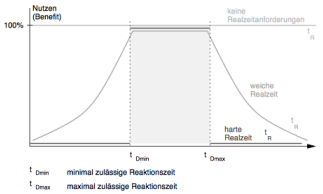
Realzeitbedingungen

- **Harte und weiche Realzeit**
- Weiches Realzeitsystem: Die 2. Realzeitbedingung (Rechtzeitigkeitsbedingung), „darf schon mal“ verletzt werden, ohne dass das gleich „eine Katastrophe“ ist.
- Hartes Realzeitsystem: Die 2. Realzeitbedingung muss unter allen Umständen und immer erfüllt werden.



Realzeitbedingungen

- **Harte und weiche Realzeit**
  - Der Unterschied lässt sich gut anhand der Nutzenfunktion verdeutlichen:
    - **Hartes Realzeitsystem**
      Ein Nutzen ist nur dann gegeben, wenn innerhalb des Zeitfensters reagiert wird.
    - **Weiches Realzeitsystem**
      Es ist auch dann ein Nutzen vorhanden, wenn die Rechtzeitigkeitsbedingung nicht grundsätzlich eingehalten wird.
    - **Kein Realzeitsystem**
      Der Nutzen ist unabhängig von den zeitlichen Anforderungen.



Realzeitbedingungen

- **Zusammenfassung Realzeitbedingungen**
  - Notwendige Bedingung: Auslastungsbedingung
  - Hinreichende Bedingung: Rechtzeitigkeitsbedingung
  - Harte Realzeit: Die Rechtzeitigkeitsbedingung muss unter allen Umständen eingehalten werden.
  - Weiche Realzeit: In zu definierenden Grenzen ist das Nichteinhalten der Rechtzeitigkeitsbedingung tolerierbar.

Systemaspekte

1. **Unterbrechbarkeit**
2. **Prioritäten**
3. **Ressourcenmanagement**

Systemaspekte

1. **Unterbrechbarkeit**
2. **Prioritäten**
3. **Ressourcenmanagement**

Systemaspekte

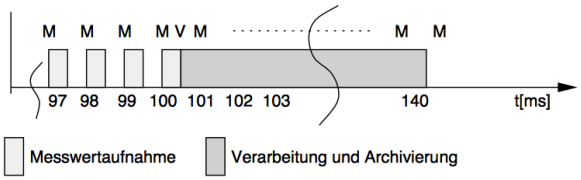
- **Unterbrechbarkeit**
  - Preemption: Eigenschaft einer Codesequenz, die Abarbeitung in mehrere Abschnitte aufteilen zu können, die dann in der korrekten Reihenfolge – aber eben mit Unterbrechungen – abgearbeitet werden können.
  - Preemption ermöglicht den Aufbau komplexer Realzeitsysteme.

Systemaspekte

- **Unterbrechbarkeit**
  - Beispiel Messwerterfassung auf einer Singlecore-Hardware:
    - Im Abstand von 1 ms müssen kontinuierliche Messwerte erfasst werden.
    - Jeweils 100 Messwerte werden am Stück weiterverarbeitet.
    - Der ausgewählte Rechner benötigt für die Erfassung eines Messwertes 0.5 ms.
    - Die Weiterverarbeitung von 100 Messwerten dauert 40 ms.
    - Damit ergibt sich eine Auslastung des Rechners von 0.5 + 0.4 = 0.9 = 90 Prozent.

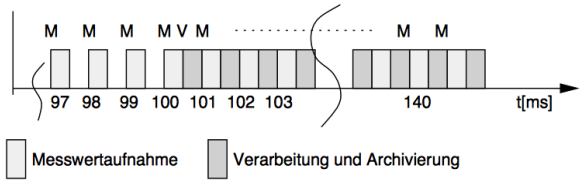
Systemaspekte

- **Unterbrechbarkeit**
  - Ohne Preemption ergibt sich bei sequenzieller Abarbeitung der Aufgaben die folgende Rechnerkernbelegung, die keine kontinuierliche Messwerterfassung ermöglicht:



Systemaspekte

- **Unterbrechbarkeit**
  - Mit Preemption müssen die beiden Aufgaben in separate Tasks (siehe Kapitel 3) aufgeteilt werden.
  - Zwischen den beiden Tasks wird eine Interprozess-Kommunikation zur Information, dass 100 Messwerte erfasst wurden, etabliert.
  - Damit ergibt sich die folgende Rechnerkernbelgung, die schritthaltende Verarbeitung ermöglicht:



Systemaspekte

1. Unterbrechbarkeit
2. **Prioritäten**
3. Ressourcenmanagement

Systemaspekte

- **Prioritäten**
  - Um schritthaltenden Betrieb zu ermöglichen, muss die Systemsoftware die Möglichkeit bieten, die Abarbeitungsreihenfolge der Codesequenzen (Tasks) festzulegen.
  - Hierfür werden im Realzeitumfeld vor allem Prioritäten eingesetzt (siehe Kapitel 3).
  - Prioritäten werden über Zahlenwerte angegeben. Wir verwenden
    - niedrige Zahl == hohe Priorität
    - hohe Zahl == niedrige Priorität
  - Der Schnittstellenstandard POSIX ordnet Zahlenwerte zu Prioritäten genau andersherum.

Systemaspekte

1. Unterbrechbarkeit
2. Prioritäten
3. **Ressourcenmanagement**

Systemaspekte

- **Ressourcenmanagement**
  - Quasi- oder real-parallel laufende Codesequenzen konkurrieren um Ressourcen (Betriebsmittel):
    - Hardware-Ressourcen, z.B. CPU, Speicher, Ein-/Ausgabegeräte
      - Diese Ressourcen werden durch Hardware und Systemsoftware verteilt.
  - Software-Ressourcen, z.B. globale Variablen
  - Kooperation: Eine Codesequenz wartet auf eine andere.
  - Konkurrenz: Codesequenzen versuchen zu beliebigen Zeitpunkten Ressourcen zu erhalten.

Systemaspekte

- **Ressourcenmanagement**
  - Beispiel „Inkonsistenz durch parallele Zugriffe“ (siehe Codebeispiel)
    - Im Eingabepuffer befindet sich die Kombination „AB“.
    - Thread 1 liest ein Zeichen ein (Zeile 5); in der Variablen ch befindet sich also der Wert 'A'.
    - Thread 2 arbeitet Zeile 5 ab und überschreibt den Wert 'A' mit 'B'.
    - Thread 1 führt Zeile 6 aus und gibt 'B' (anstatt 'A') aus.
    - Thread 2 führt Zeile 6 aus und gibt ebenfalls 'B' aus.

```
1: char ch; // Variable ist global
2:
3: void echo()
4: {
5:     ch = getchar(); // Zeichen einlesen
6:     putchar( ch );  // Zeichen ausgeben
7: }
```

Systemaspekte

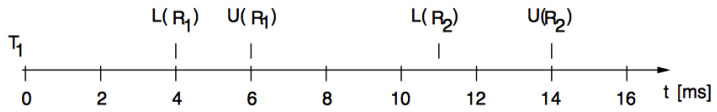
- **Ressourcenmanagement**
  - Codesequenzen, die den Zugriff auf gemeinsam genutzte Betriebsmittel implementieren, werden mit „kritische Abschnitte“ bezeichnet.
  - Damit es durch die kritischen Abschnitte zu keinen Inkonsistenzen kommt, werden Schutzmechanismen benötigt: Semaphor, Mutex, Spinlocks, Interruptsperre, Monitor (siehe Kapitel 4).
  - Diese realsieren meist einen „gegenseitigen Ausschluss“ (mutal exclusion):
    - Zu einem Zeitpunkt darf immer nur eine Codesequenz den kritischen Abschnitt abarbeiten.
    - Es findet eine Sequenzialisierung statt.

Systemaspekte

- **Ressourcenmanagement**
  - Ressourcen werden mit einem Buchstaben und eventuell einem Index gekennzeichnet:
    - R (Ressource)
    - S (Semaphor)
  - Der Programmierer kennzeichnet mit Funktionsaufrufen den Anfang und das Ende eines kritischen Abschnitts:
    - L(R) – Beginn des kritischen Abschnitts – Lock
    - U(R) – Ende des kritischen Abschnitts – Unlock

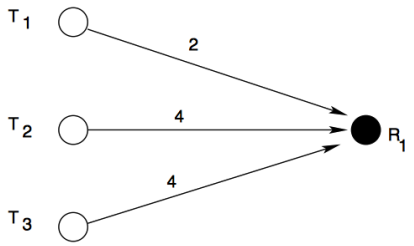
Systemaspekte

- **Ressourcenmanagement**
  - Die zeitliche Länge eines kritischen Abschnitts wird mit $t_{CS,R}$ bezeichnet.
  - In der Abbildung sind die Längen der beiden kritischen Abschnitte
  - $t_{CS,R1}$ = 2 ms
  - $t_{CS,R2}$ = 3 ms



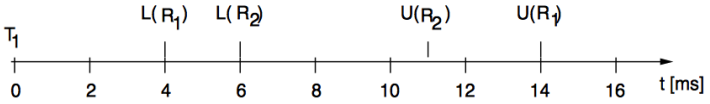
Systemaspekte

- **Ressourcengraph: Verschiedene Tasks benutzen die gleiche Ressource**



Systemaspekte

- **Ressourcenmanagement**
  - Komplizierter wird es, wenn Ressourcen „verschränkt“ verwendet werden.
  - Die Länge der „äußeren“ Ressource beinhaltet die Länge der „inneren“:
  - $t_{CS,R1}$ = 10 ms
  - $t_{CS,R2}$ = 5 ms



Systemaspekte

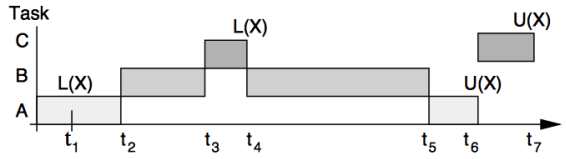
- **Ressourcenmanagement**
  - Der Schutz kritischer Abschnitte führt zu einem geänderten Zeitverhalten des Realzeitsystems.
  - Für die notwendige, zeitliche Analyse (Realzeitnachweis) werden bei Realzeitsystemen, die auf Prioriäten beruhen, die folgenden Parameter benötigt (siehe Kapitel 8):
    - $t_{CS,R}$: Zeitliche Länge des kritischen Abschnitts R
    - $\Pi(R)_i$: Priorität der Ressource R. Diese entspricht der höchsten Priorität der Task, die die Ressource verwenden.
    - Π_s: Priorität des Gesamtsystems, diese entspricht der höchsten Priorität aller Ressourcen.

Systemaspekte

- **Ressourcenmanagement**
  - Prioritätsinversion: Hierunter versteht man die Situation, dass eine Task warten muss, obwohl sie die höchste Priorität im System hat. Das ist beispielsweise dann der Fall, wenn sie einen kritischen Abschnitt betreten möchte, der gerade von einer Task mit niedrigerer Priorität bearbeitet wird.
  - Verschärft wird die Situation durch mittelpriore Tasks, die die niedrigpriore und damit vor allem die hochpriore noch weiter verzögern.

Systemaspekte

- **Prioritätsinversion**



t_1	Task A belegt die Ressource X.
t_2	Task B (mit höherer Priorität) wird rechenbereit.
t_3	Task C (mit der höchsten Priorität) wird rechenbereit.
t_4	Task C versucht, die Ressource X zu belegen, und wird schlafen gelegt. Task B hat von den rechenbereiten Prozessen die höchste Priorität.
t_5	Nach Beendigung von B kann A weiterarbeiten.
t_6	Task A gibt die Ressource frei, erst jetzt kann C weiterarbeiten.

Systemaspekte

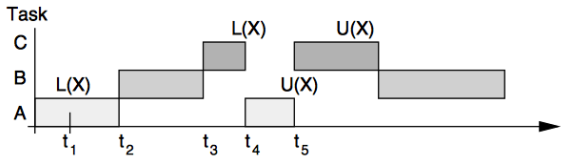
- **Zur Entschärfung der zeitlichen Verzögerungen bieten sich verschiedene Methoden an:**
  - Unterbrechungssperre (NPCS)
  - Prioritätsvererbung (PIP)
  - Priority Ceiling (PCP)

Systemaspekte

- **Non Preemptible Critical Sections (NPCS)**
  - Reduzierung der zeitlichen Verzögerungen durch eine Unterbrechungssperre (NPCS).
  - Einfach zu implementieren (z.B. Interruptsperre)
  - Deadlocks können nicht auftreten
  - gilt für statische und dynamische Prioritäten (später mehr dazu)
  - Nachteil?

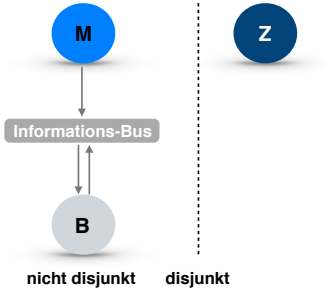
Systemaspekte

- **Prioritätsvererbung (PIP)**



t_1	Task A belegt die Ressource X (Lock).
t_2	Task B (mit höherer Priorität) wird rechenbereit.
t_3	Task C (mit der höchsten Priorität) wird rechenbereit.
t_4	Task C versucht, die Ressource X zu belegen, und wird schlafen gelegt.
t_4	Task A erbt die Priorität von Task C und wird rechenbereit.
t_5	Task A gibt X wieder frei und bekommt die ursprüngliche Priorität.

Beispiel: Mars Pathfinder



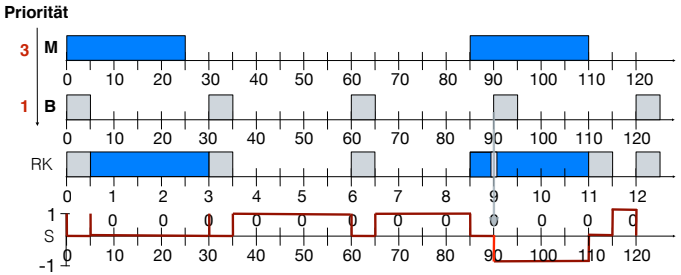
Z Zustandsüberwachung des Realzeitsystems

M Erfassung Messwerte

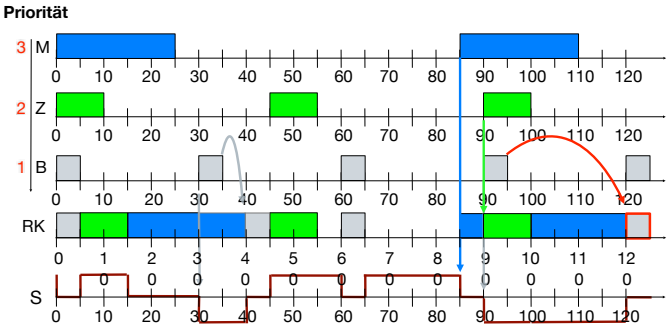
B Datenaustausch über Softwarebus

Zugriff auf Informationsbus über **Semaphore**

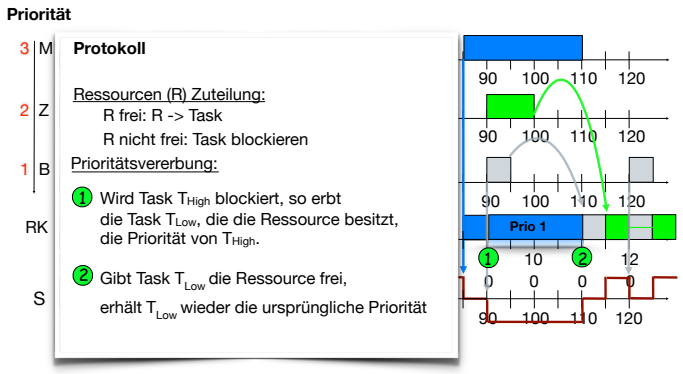
2 Tasks



Alle 3 Tasks



Priority Inhertiance Protokoll



Systemaspekte

- **Prioritätsvererbung (PIP)**
- relativ einfach zu implementieren (keine Informationen vom Programmierer nötig)
- Arten des Blockierens:
  - direkt
  - indirekt (auch für alle anderen Tasks!), aufgrund der Vererbung
- verhindert keine Deadlocks

Systemaspekte

- **Priority Ceiling (PCP)**
- Bedingung: Die von den Tasks benötigten Ressourcen sind vor dem Start bekannt!
- «Priority Ceiling» ("Prioritäten Obergrenze") wird im System berechnet.
- Die Priority Ceiling einer jeden Ressource R ist die Priorität der höchstprioren Task, die diese Ressource benötigt.
- Die aktuelle Priority Ceiling Πs des Systems entspricht der höchsten Priorität der Priority Ceiling der Ressourcen, die gerade von Tasks benutzt werden.

Systemaspekte

- **Priority Ceiling (PCP): Zuteilungsregel**
- Wenn eine Task T zur Zeit t eine Ressource anfordert, und R nicht verfügbar ist, wird die Anforderung abgelehnt und T blockiert bis R frei ist:
  - erhält T die Ressource wenn T's Priorität π(t) höher als Πs(t) ist.
- Falls T's Priorität π(t) nicht höher als Πs(t) ist
  - erhält T die Ressource R nur, wenn T die Task ist, deren Ressourcen Zugriffe die derzeitige Πs(t) ausgelöst hat. Ansonsten wird die Ressourcen Anforderung abgelehnt und T wird blockiert.
- Prioritätenvererbung an Tasks wie bei PIP

Systemaspekte

- **Zusammenfassung Systemaspekte**
  - Preemption – Wir fordern, dass Jobs unterbrechbar sind.
  - Prioritäten – Die Abarbeitungsreihenfolge kann festgelegt werden (beispielsweise über Prioritäten).
  - Kritische Abschnitte werden per L(R) und U(R) geschützt.
  - Der Schutz kritischer Abschnitte führt zu Zeitanomalien, welche durch den Einsatz von Protokollen deterministisch ist

Zusammenfassung Realzeitbetrieb

Zusammenfassung Realzeitbetrieb

- Die Außenwelt ist im Wesentlichen gekennzeichnet durch
  - Rechenzeitanforderungen
  - Minimale Prozesszeiten
  - Minimal zulässige Reaktionszeiten
  - Maximal zulässige Reaktionszeiten
- Das Taskset ist im Wesentlichen gekennzeichnet durch
  - Minimale Verarbeitungszeit
  - Maximale Verarbeitungszeit
  - Minimale Reaktionszeit
  - Maximale Reaktionszeit

Zusammenfassung Realzeitbetrieb

- Wir gehen davon aus, dass das System folgendes ermöglicht
  - Preemption
  - Prioritäten (oder andere Mechanismen zur Festlegung der Abarbeitungsreihenfolge)
  - Ressourcenmanagement (zum Schutz kritischer Abschnitte)
- Für die schritthaltende Verarbeitung müssen die zwei Realzeitbedingungen erfüllt sein:
  - Die maximale Gesamtauslastung muss kleiner oder gleich der Anzahl der CPU-Kerne sein.
  - Die minimale und maximale Reaktionszeit auf eine Rechenzeitanforderung muss innerhalb des Zeitfensters liegen.

Zusammenfassung Realzeitbetrieb

- Harte Realzeit: Die Rechtzeitigkeitsbedingung wird unter allen Umständen eingehalten.
- Weiche Realzeit: Eine Deadline-Verletzung darf schon mal vorkommen...
- Zum Schutz kritischer Abschnitte werden Lock-Primitive (z.B. Semaphor) eingesetzt. Der Einsatz führt zu Zeitanomalien, beispielsweise zur Prioritätsinversion.
- Geeignete Protokolle erlauben einen zeitlichen Determinismus beim Umgang mit Ressourcen.