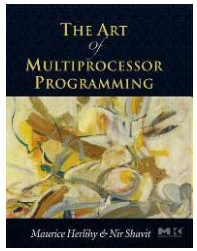
Kapitel 3

Nebenläufigkeit

Inhalt

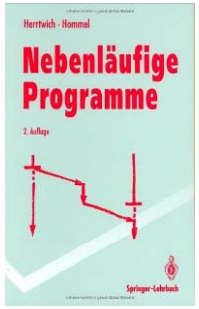
- Motivation
- Unterbrechungen (Interrupts)
- (Software-) Prozesse
- Threads
- Interprozesskommunikation (IPC)

Literatur



Maurice Herlihy, Nir Shavit,
The Art of Multiprocessor
Programming, 2008

A.S.Tanenbaum, Moderne
Betriebssysteme, 2002



R.G.Herrtwich, G.Hommel,
Nebenläufige Programme
1998

- Edward Lee: The Problem with Threads:
  http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.pdf
- http://www.beyondlogic.org/interrupts/interupt.htm
- http://www.llnl.gov/computing/tutorials/pthreads/

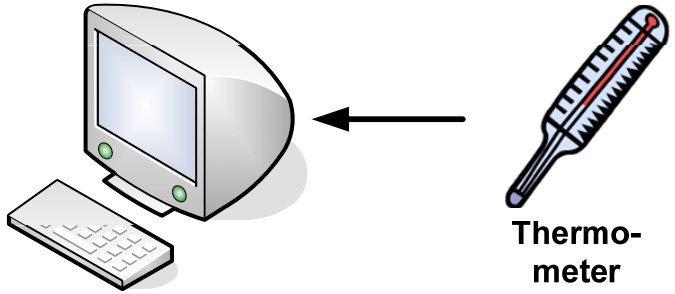
Definition von Nebenläufigkeit

- **Allgemeine Bedeutung**: Nebenläufige Ereignisse sind nicht kausal abhängig. Ereignisse (bzw. Ereignisfolgen) sind dann nebenläufig, wenn keines eine Ursache im anderen hat.
- **Bedeutung in der Informatik**: Nebenläufig bezeichnet hier die Eigenschaft von Programmcodes, nicht linear hintereinander ausgeführt werden zu müssen, sondern zeitlich parallel zueinander ausführbar zu sein.
- Aktionen (Programmschritte) können parallel (gleichzeitig oder quasi gleichzeitig) ausgeführt werden, wenn keine das Resultat der anderen benötigt. Die parallele Ausführung von mehreren unabhängigen *Prozessen* (siehe später) auf einem oder mehreren Prozessoren bezeichnet man als *Multitasking*. Die parallele Ausführung von Teilsequenzen innerhalb eines Prozesses heißt *Multithreading*.

Motivation

- Gründe für nebenläufige Ausführung von Programmen in Echtzeitsystemen:
  - Echtzeitsysteme sind häufig verteilte Systeme (Systeme mit mehrere Prozessoren).
  - Zumeist werden zeitkritische und zeitunkritische Aufgaben parallel berechnet.
  - Bei reaktiven Systemen ist die maximale Antwortzeit häufig limitiert.
  - Abbildung der parallelen Abläufe im technischen Prozeß
- Aber: kleinere (Monoprozessor-)Echtzeit-Systeme verzichten häufig auf die parallele Ausführung von Code, weil der Aufwand für die Prozeßverwaltung zu hoch ist.
  Dennoch auch hier: typischerweise Parallelverarbeitung in „Hauptprogramm“ und „Unterbrechungsbehandler“ (interrupt service routine, interrupt handler)

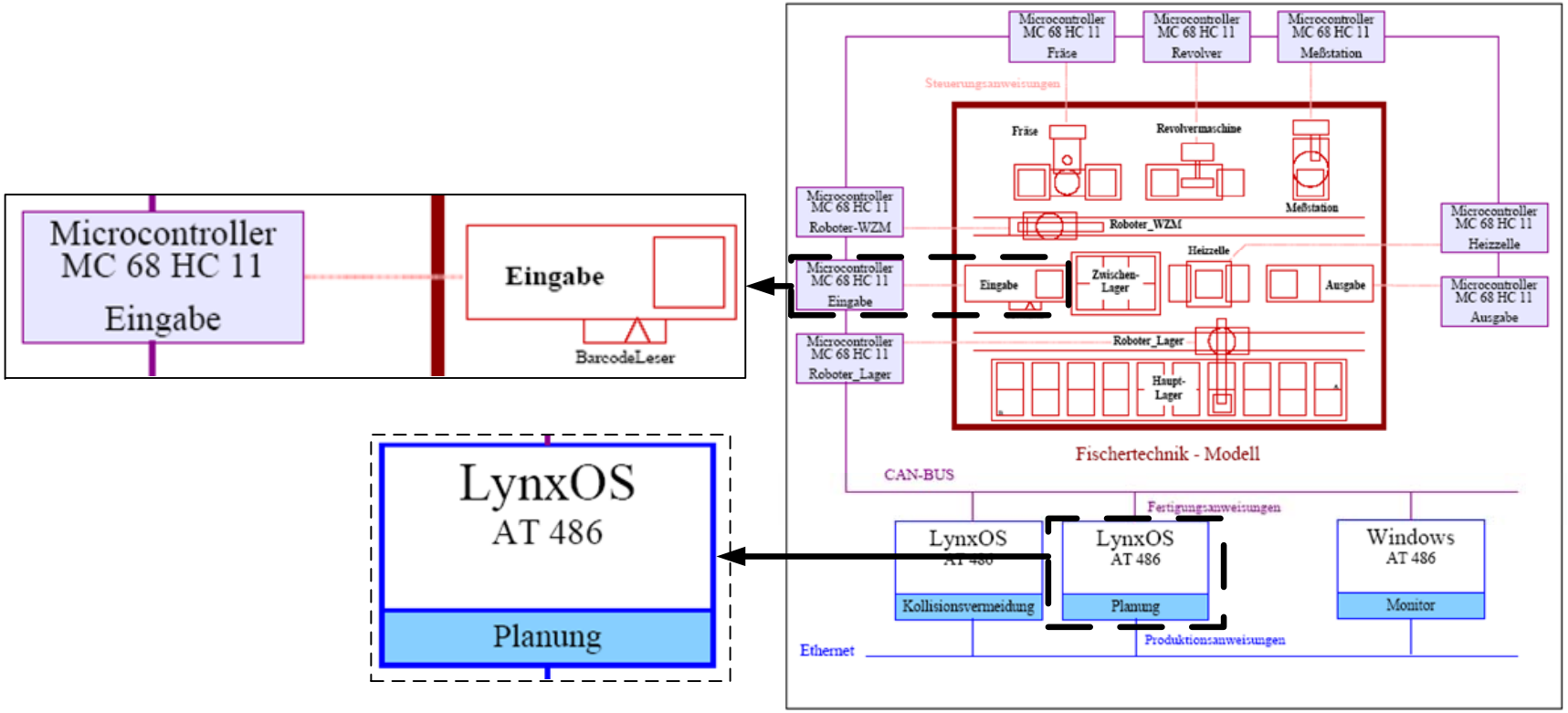
Anwendungsfälle für Nebenläufigkeit (Unterbrechungen)



Signal falls Temperaturwert überschritten wird
⇨ **Unterbrechungen (interrupts)**

Allgemeines Anwendungsgebiet: hauptsächlich zur Anbindung von externer Hardware

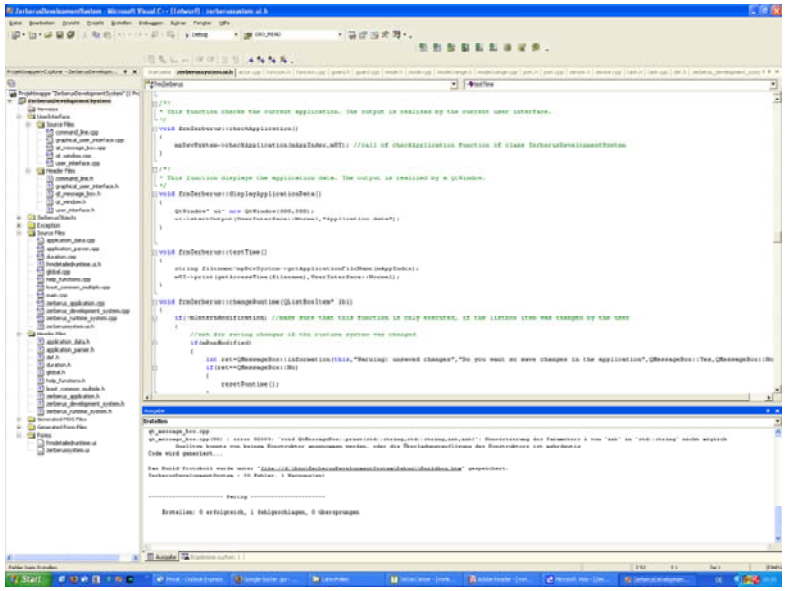
Anwendungsfälle für Nebenläufigkeit (Prozesse)



Verteiltes System zur Steuerung der Industrieanlage ⇨ **Prozesse (tasks)**

Allgemeine Anwendungsgebiete: verteilte Systeme, unterschiedlichen Anwendungen auf einem Prozessor

Anwendungsfälle für Nebenläufigkeit (Threads)



Reaktion auf Nutzereingaben trotz Berechnungen (z.B. Übersetzen eines Programms)

⇨ **leichtgewichtige Prozesse (Threads)**

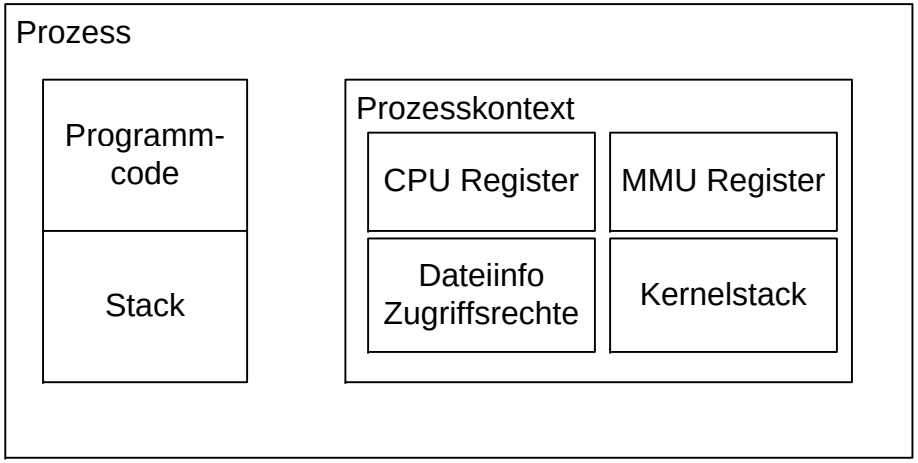
Allgemeines Anwendungsgebiet: unterschiedliche Berechnungen im gleichen Anwendungskontext

Nebenläufigkeit

Prozesse

Definition

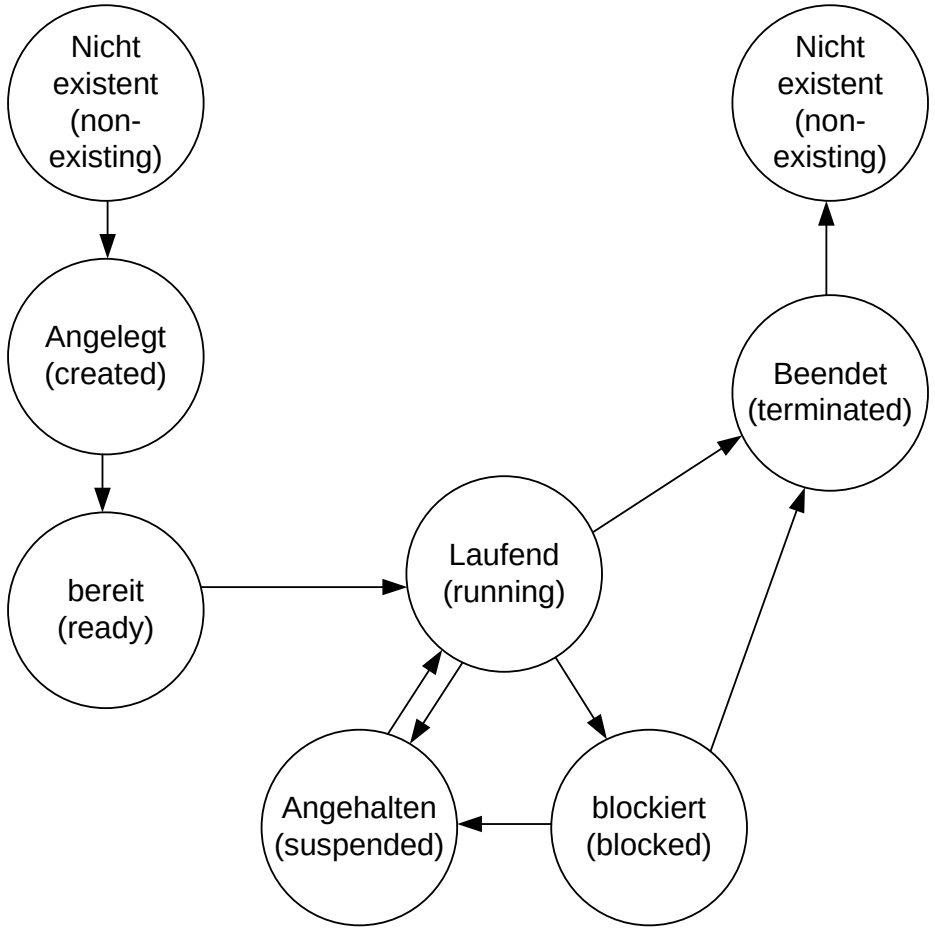
- **Prozess**: Abstraktion eines sich in Ausführung befindlichen Programms
- Die gesamte Zustandsinformation der Betriebsmittel für ein Programm wird als eine Einheit angesehen und als Prozess bezeichnet.
- Prozesse können weitere Prozesse erzeugen ⇨ Vater-,Kinderprozesse.



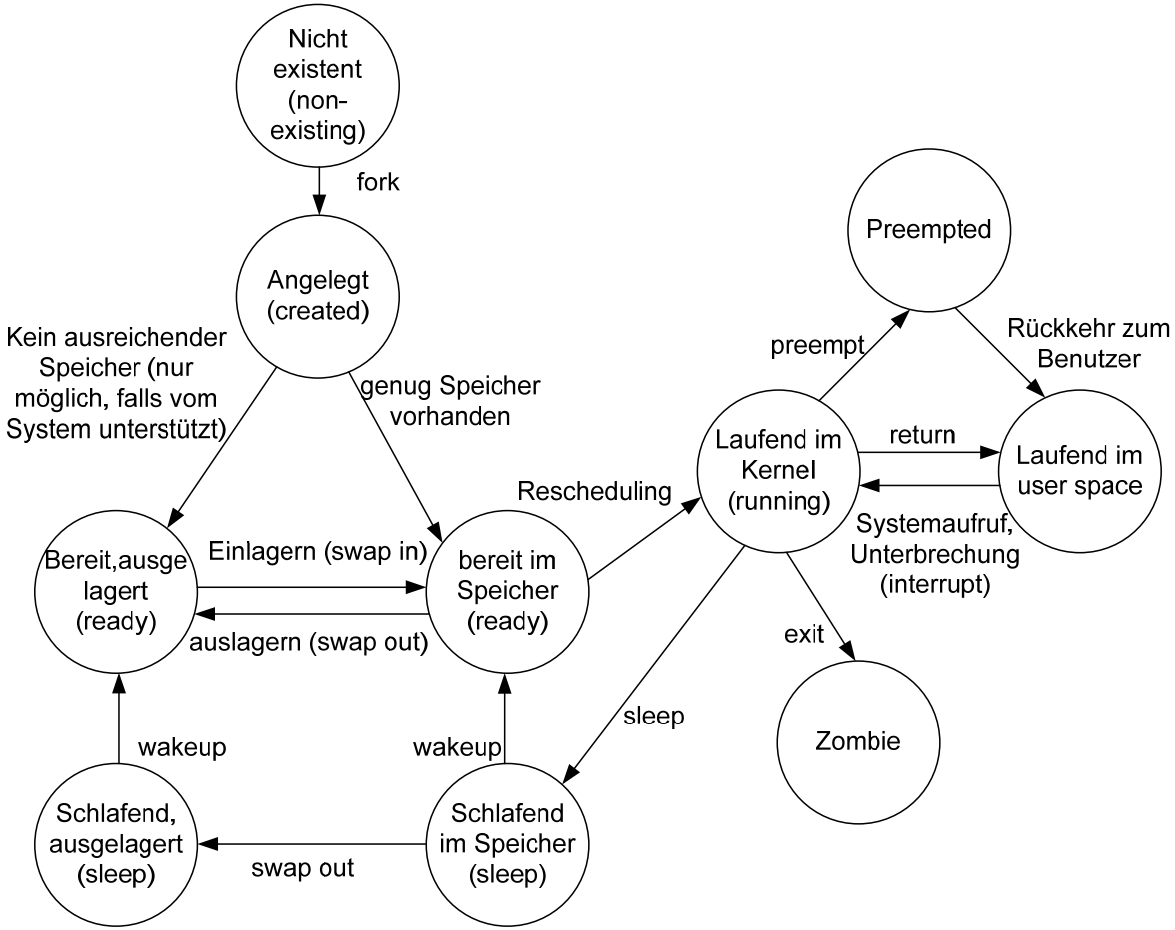
Prozessausführung

- Zur Prozessausführung werden diverse Resourcen benötigt, u.a.:
  - Prozessorzeit
  - Speicher
  - sonstige Betriebsmittel (z.B. spezielle Hardware)
- Die Ausführungszeit ist neben dem Programm abhängig von:
  - Leistungsfähigkeit des Prozessors
  - Verfügbarkeit der Betriebsmittel
  - Eingabeparametern
  - Verzögerungen durch andere (wichtigere) Aufgaben

Prozesszustände (allgemein)



Prozeßzustände in Unix



Fragen bei der Implementierung

- Welche Betriebsmittel sind notwendig?
- Welche Ausführungszeiten besitzen einzelne Prozesse?
- Wie können Prozesse kommunizieren?
- Wann soll welcher Prozess ausgeführt werden?
- Wie können Prozesse synchronisiert werden?

Klassifikation von Prozessen

- periodisch vs. aperiodisch
- statisch vs. dynamisch
- Wichtigkeit der Prozesse (kritisch, notwendig, nicht notwendig)
- speicherresident vs. verdrängbar
- Prozesse können auf
  - einem Rechner (Pseudoparallelismus)
  - einem Multiprozessorsystem mit Zugriff auf gemeinsamen Speicher
  - oder auf einem Multiprozessorsystem ohne gemeinsamen Speicher

  ausgeführt werden.

Nebenläufigkeit

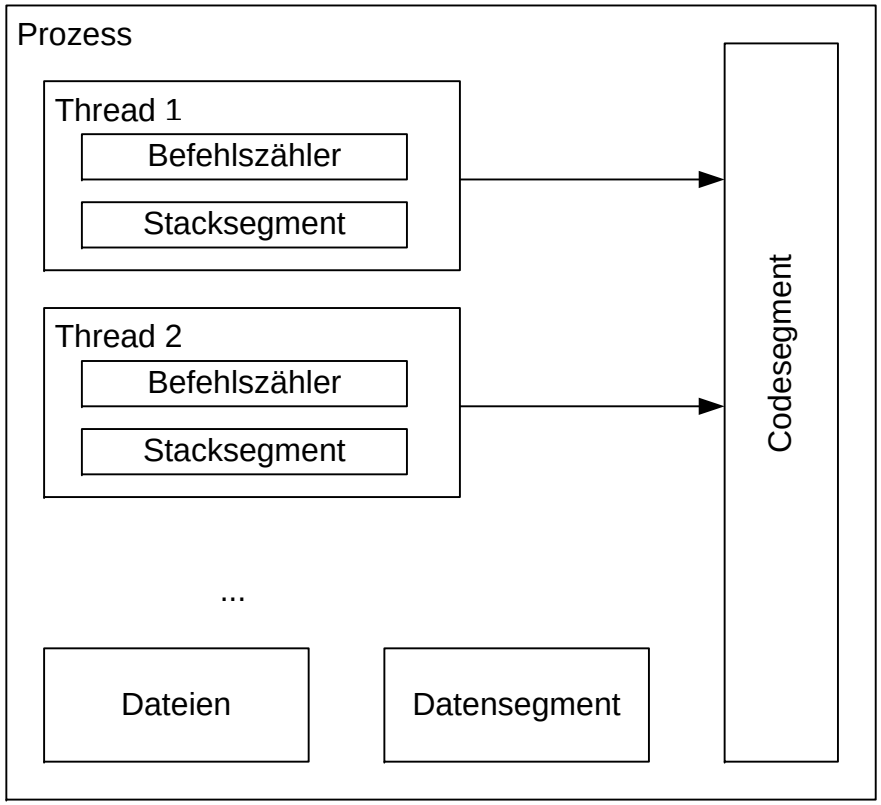
Threads

Leichtgewichtige Prozesse (Threads)

- Der Speicherbedarf von Prozessen ist in der Regel groß (CPU-Daten, Statusinformationen, Angaben zu Dateien und EA-Geräten...).
- Bei Prozesswechsel müssen die Prozessdaten ausgetauscht werden $\Rightarrow$ hohe Systemlast, zeitaufwendig.
- Viele Systeme erfordern keine komplett neuen Prozesse.
- Vielmehr sind Programmabläufe nötig, die auf den gleichen Prozessdaten arbeiten.

⇨ Einführung von Threads

Threads


Prozess
Thread 1
Befehlszähler
Stacksegment
Thread 2
Befehlszähler
Stacksegment
...
Dateien
Datensegment
Codesegment

Prozesse vs. Threads

- Verwaltungsaufwand von Threads ist deutlich geringer
- Effizienzvorteil: bei einem Wechsel von Threads im gleichen Prozessraum ist kein vollständiger Austausch des Prozesskontextes notwendig.
- Kommunikation zwischen Threads des gleichen Prozesses kann über gemeinsamen Speicher erfolgen.
- Zugriffe auf den Speicherbereich anderer Prozesse führen zu Fehlern.
- Probleme bei Threads: durch die gemeinsame Nutzung von Daten kann es zu Konflikten kommen.

Nebenläufigkeit

Unterbrechungen

Binding Rechnersystem-Umwelt

- Es muss ein Mechanismus gefunden werden, der es erlaubt, Änderungen der Umgebung (z.B. Druck einer Taste) zu registrieren.
- **1. Ansatz**: Abfrage (Polling)
  Es werden die E/A-Register reihum nach Änderungen abgefragt und bei Änderungen spezielle Antwortprogramme ausgeführt.
  - Vorteile:
    - bei wenigen EA-Registern sehr kurze Latenzzeiten
    - bei einer unerwarteten Ereignisflut wird das Zeitverhalten des Programms nicht übermäßig beeinflusst
    - Kommunikation erfolgt synchron mit der Programmausführung
  - Nachteile:
    - die meisten Anfragen sind unnötig
    - hohe Prozessorbelastung
    - Reaktionszeit steigt mit der Anzahl an Ereignisquellen

Lösung: Einführung des Begriffs der Unterbrechung

- **2. Ansatz**: Unterbrechung (Interrupt)
- Eine Unterbrechung stoppt die Verarbeitung des laufenden Programms. Die Wichtigkeit des Ereignisses, welches die Unterbrechung ausgelöst hat, wird überprüft. Darauf basierend erfolgt die Entscheidung, welche Reaktion erfolgt.
- Vorteile:
  - Prozessorressourcen werden nur dann beansprucht, wenn es nötig ist
- Nachteile:
  - Nicht-Determinismus: Unterbrechungen asynchron zum Programmlauf (und zum Prozessorzustand) eintreffen.

Unterbrechungen

- **Unterbrechungen**: Stopp des Hauptprogrammablaufs, Aufnahme der Programmausführung eines „Unterbrechungsbehandlers (UBB)“ an einer anderen Stelle; nach Beendigung des UBB (zumeist) Rückkehr an die Stelle des Auftritts der Unterbrechung im Hauptprogramm.
- **Synchrone** Unterbrechungen: treten, falls sie auftreten, immer an *derselben Stelle* im Programmcode auf. Man bezeichnet sie auch als *Traps* oder *Exceptions* bzw. „Software-Interrupts“
- **Asynchrone** Unterbrechungen: Auftrittszeitpunkt ist unbestimmt; es kann nicht gesagt werden, an welcher Stelle der Hauptprogrammausführung der Prozessor zum Zeitpunkt der Unterbrechung ist. Asynchrone Unter-brechungen werden auch als Interrupts bezeichnet; weil sie von der Hardware-Peripherie erzeugt werden, auch als *Hardware-Interrupts*. Sie üben „Brückenfunktion“ zwischen Hardware und Software aus.

Synchrone Unterbrechungen (Traps/Exceptions)

- Werden durch das Programm selbst ausgelöst, d.h. dasselbe Programm, ausgeführt mit denselben Parametern wird in der Regel an derselben Stelle dieselbe Unterbrechung auslösen (vorhersagbar in dieselbe „Falle“ laufen)
- **Auslösung bei Fehler** – Ausnahme/Exception, Beispiele:
  - Arithmetikfehler (Division by zero, overflow, not-a-number NaN, …)
  - Speicherfehler (Page Fault, segment Fault, memory full, …)
  - Befehlsfehler (Illegal instruction, privileged instruction, bus error, …)
  - Peripheriefehler (End-of-file EOF, channel blocked, unknown device, …)
- Bei Exceptions **nur dann** Rückkehr an den Auftrittspunkt, wenn die Fehlerbedingung im Ausnahmebehandler beseitigt werden kann, andernfalls Abbruch (resumption vs. termination)
- **Auslösung durch spezifische Instruktion**: Breakpoint, SWI, TRAP, INT, … entweder zum Zwecke des „Debuggings“ oder zum Aufruf von Betriebssystem-Diensten (z.B. MS-DOS „INT 21h“, siehe z.B. http://en.wikipedia.org/wiki/MS-DOS_API)
- Traps können auch benutzt werden, um einen Hardware-Interrupthandler zu testen.