



Kapitel 5

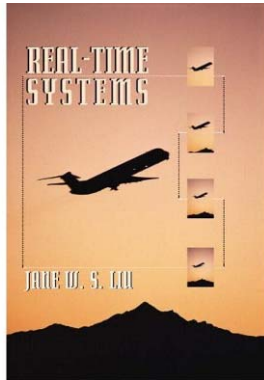
Echtzeitbetriebssysteme



Inhalt

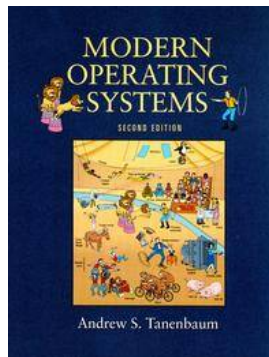
- Grundlagen
- Betrachtung diverser Betriebssysteme:
 - Domänenspezifische Betriebssysteme:
 - OSEK
 - TinyOS
 - Klassische Echtzeitbetriebssysteme
 - QNX
 - VxWorks
 - PikeOS
 - Linux- / Windows-Echtzeitvarianten
 - RTLinux/RTAI
 - Linux Kernel 2.6
 - Windows CE

Literatur



Jane W. S. Liu, Real-Time Systems, 2000

Dieter Zöbel, Wolfgang Albrecht:
Echtzeitsysteme: Grundlagen und Techniken, 1995



Andrew S. Tanenbaum: Modern Operating Systems, 2001

Arnd Heursch et al.: Time-critical tasks in Linux 2.6, 2004

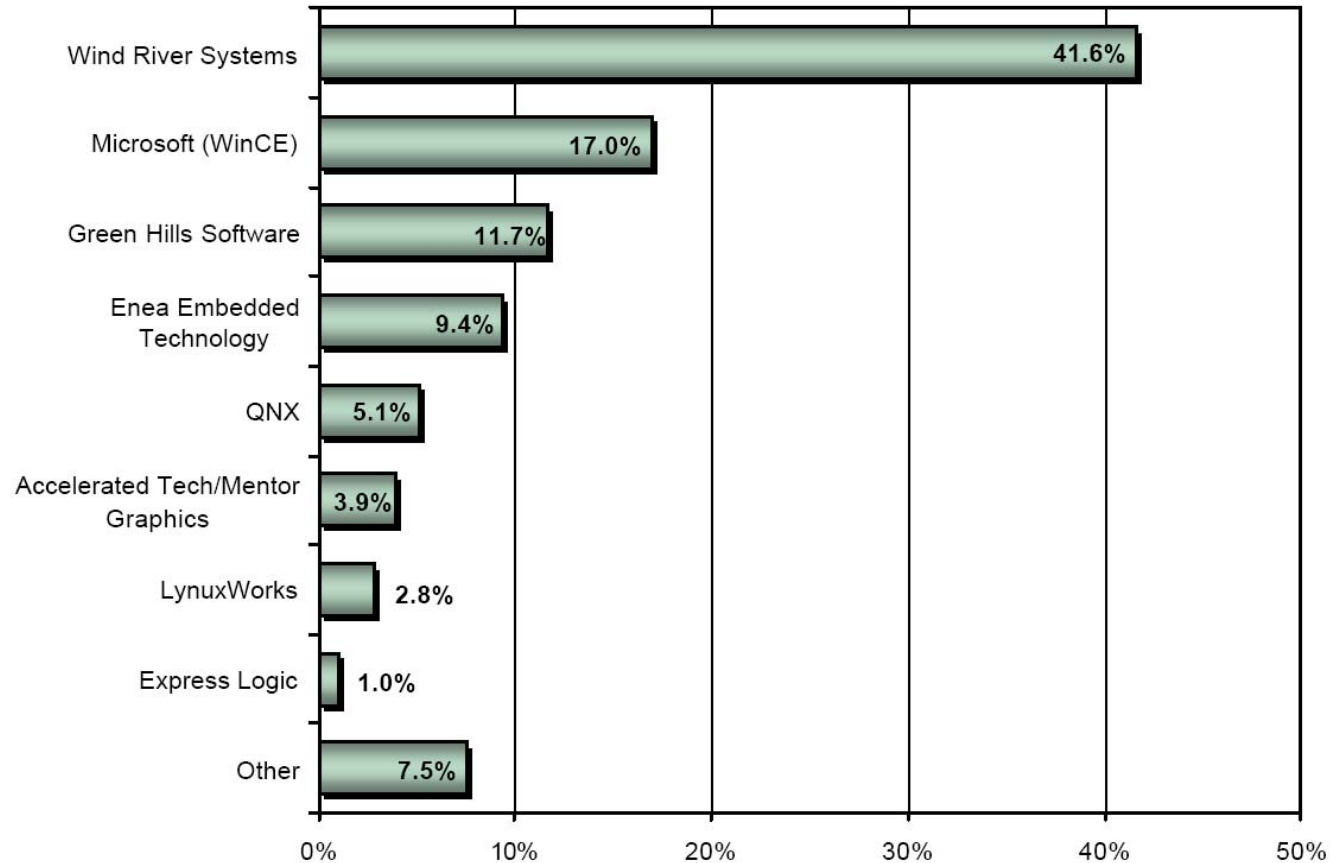
http://inf3-www.informatik.unibw-muenchen.de/research/linux/hannover/automation_conf04.pdf



Interessante Links

- <http://www.mnis.fr/en/support/doc/rtos/>
- <http://aeolean.com/html/RealTimeLinux/RealTimeLinuxReport-2.0.0.pdf>
- <http://www.osek-vdx.org/>
- <http://www.qnx.com/>
- <http://www.windriver.de>
- <http://www.fsmlabs.com>
- <http://www.rtai.org>
- <http://www.tinyos.net/>

Marktaufteilung (Stand 2004)



Marktanteil am Umsatz, Gesamtvolumen 493 Mio. Dollar, Quelle: The Embedded Software Strategic Market Intelligence Program 2005



Anforderungen an Echtzeitbetriebssysteme

- Echtzeitbetriebssysteme unterliegen anderen Anforderungen als Standardbetriebssysteme:
 - stabiler Betrieb rund um die Uhr
 - definierte Reaktionszeiten
 - parallele Prozesse
 - Unterstützung von Mehrprozessorsystemen
 - schneller Prozesswechsel (geringer Prozesskontext)
 - echtzeitfähige Unterbrechensbehandlung
 - echtzeitfähiges Scheduling
 - echtzeitfähige Prozesskommunikation
 - umfangreiche Zeitdienste (absolute, relative Uhren, Weckdienste)
 - einfaches Speichermanagement



Fortsetzung

- Unterstützung bei der Ein- und Ausgabe
 - vielfältigste Peripherie
 - direkter Zugriff auf Hardware-Adressen und -Register durch den Benutzer
 - Treiber in Benutzerprozessen möglichst schnell und einfach zu implementieren
 - dynamisches Binden an den Systemkern
 - direkte Nutzung DMA
 - keine mehrfachen Puffer: direkt vom Benutzerpuffer auf das Gerät
- Einfachste Dateistrukturen
- Protokoll für Feldbus oder LAN-Bus, möglichst hardwareunterstützt
- Aufteilung der Betriebssystemfunktionalität in optionale Komponenten (Skalierbarkeit)



Echtzeitbetriebssysteme

Kriterien zur Beurteilung



Beurteilung von Echtzeitbetriebssystemen

- Folgende Aspekte werden wir genauer betrachten:
 - Schedulingverfahren
 - Prozessmanagement
 - Speicherbedarf (Footprint)
 - Garantierte Reaktionszeiten



Schedulingverfahren

- Fragestellung:
 - Welche Konzepte sind für das Scheduling von Prozessen verfügbar?
 - Gibt es Verfahren für periodische Prozesse?
 - Wie wird dem Problem der Prioritätsinversion begegnet?
 - Wann kann eine Ausführung unterbrochen werden?



Arten von Betriebssystemen

- Betriebssysteme werden in drei Klassen unterteilt:
 - Betriebssysteme mit **kooperativen Scheduling**: es können verschiedene Prozesse parallel ausgeführt werden. Der Dispatcher kann aber einem Prozess den Prozessor nicht entziehen, vielmehr ist das Betriebssystem auf die Kooperation der Prozesse angewiesen (z.B. Windows 95/98/ME)
 - Betriebssysteme mit **präemptiven Scheduling**: einem laufenden Prozess kann der Prozessor entzogen werden, falls sich der Prozess im Userspace befindet. (z.B. Linux, Windows 2000/XP)
 - **Präemptible Betriebssysteme**: der Prozessor kann dem laufenden Prozess jederzeit entzogen werden, auch wenn sich dieser im Kernelkontext ausgeführt wird.

⇒ Echtzeitsysteme müssen präemptibel sein.



Prozessmanagement

- Bewertung eines Betriebssystems nach:
 - Beschränkung der Anzahl von Prozessen
 - Möglichkeiten zur Interprozesskommunikation
 - Kompatibilität der API mit Standards (z.B. POSIX) zur Erhöhung der Portabilität



Speicherbedarf

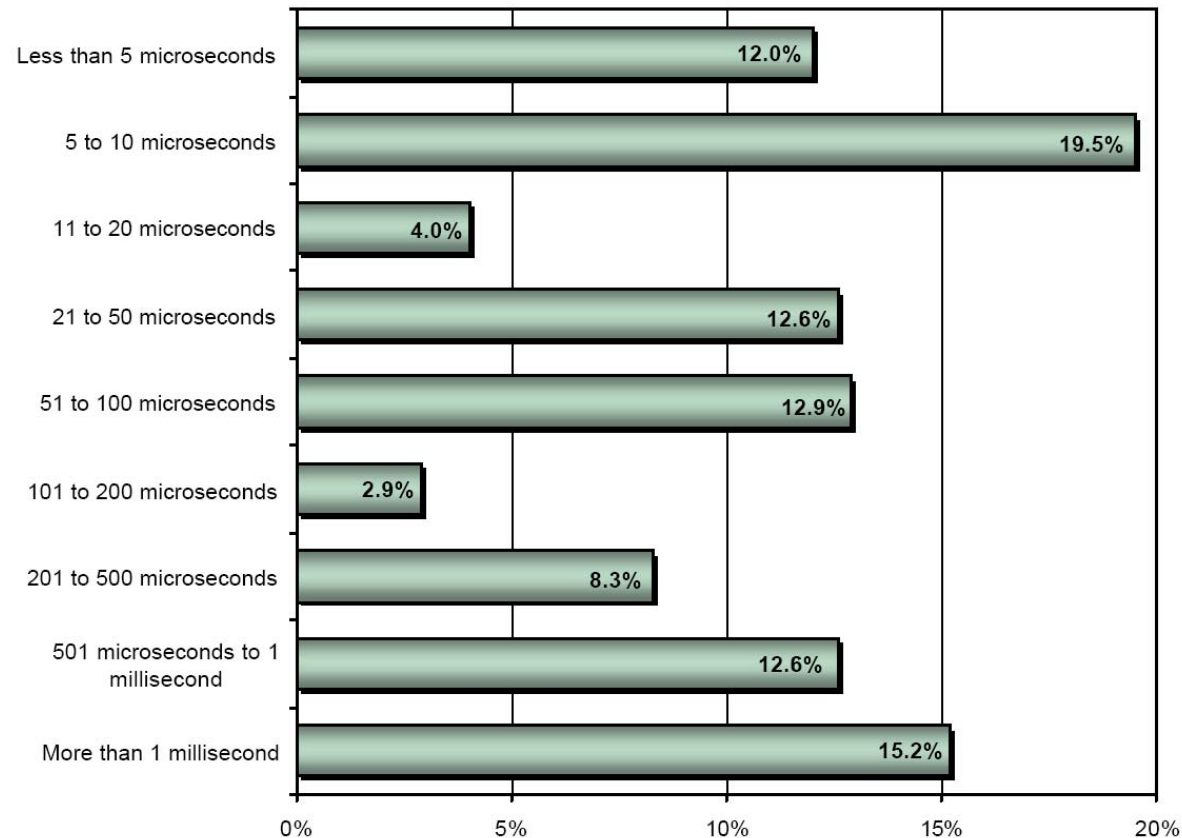
- Echtzeitbetriebssysteme werden auf sehr unterschiedlicher Hardware ausgeführt
 - Der verfügbare Speicher variiert sehr stark.
 - Typische Betriebssystemfunktionalitäten (z.B. Dateisysteme, graphische Oberfläche) werden oft gar nicht benötigt.
- ⇒ Echtzeitsysteme müssen aus diesen Gründen skalierbar sein:
 - Möglichkeit zur Auswahl einzelner Module entsprechend den Anforderungen an die Funktionalität der Anwendung.
 - Entscheidend ist der **minimale Speicherbedarf (Footprint)**.



Reaktionszeiten

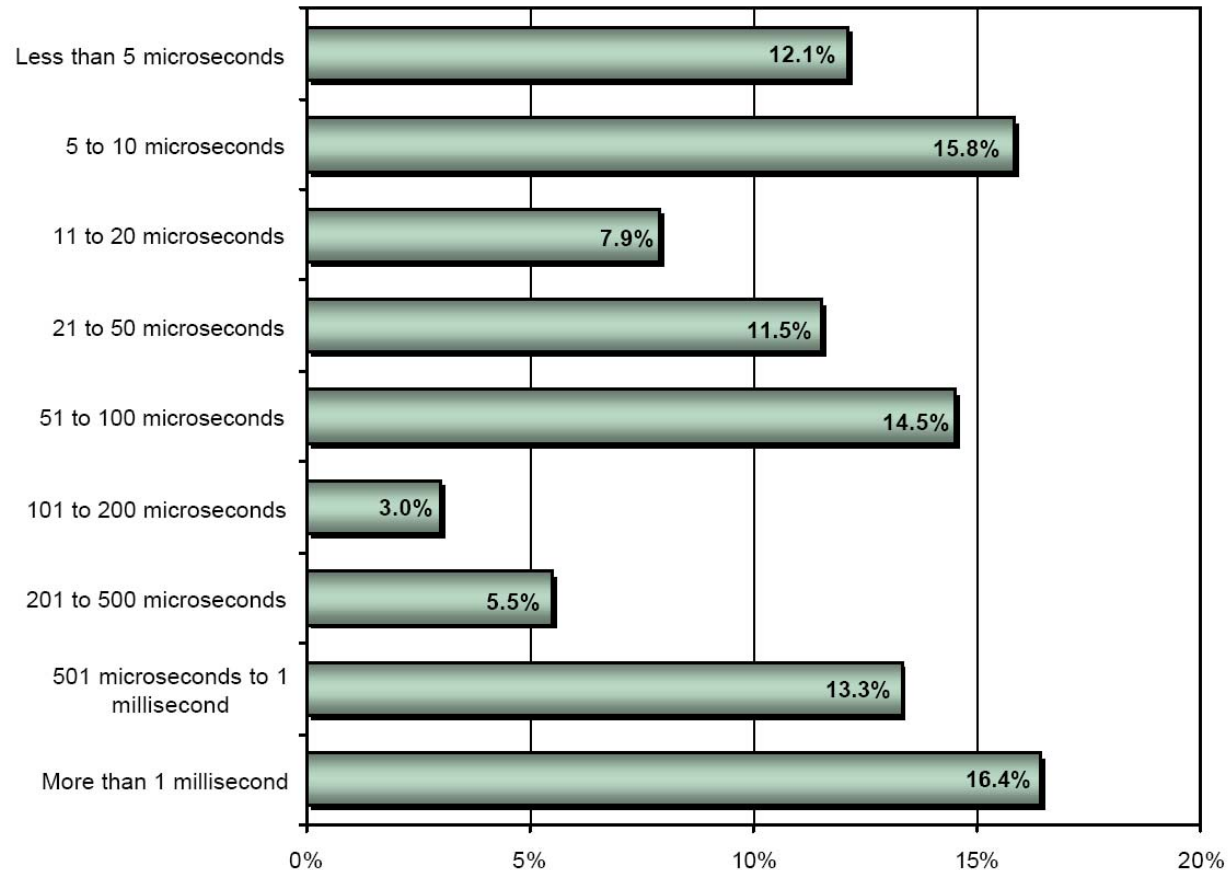
- Die Echtzeitfähigkeit wird durch die Messung folgender Zeiten bestimmt:
 - **Unterbrechungsantwortzeiten (interrupt latency)**: der Zeitraum zwischen dem Auftreten einer Unterbrechung und der Ausführung des ersten Befehls der dazugehörigen Unterbrechungsbehandlungsroutine
 - **Schedulinglatenz (scheduling latency)**: Zeit von der Ausführung des letzten Befehls des Unterbrechungsbehandlers bis zur Ausführung der ersten Instruktion des Prozesses, der durch das Auftreten der Unterbrechung in den bereiten Zustand wechselt.
 - Zeiten für einen **Kontextwechsel (context switch latency)**: Zeit von der Ausführung des letzten Befehls eines Prozesses im Userspace bis zur Ausführung der ersten Instruktion des nächsten Prozesses im Userspace.

Anforderungen an Unterbrechungsantwortzeiten



Typische Anforderungen an Antwortzeiten, Quelle: The Embedded Software Strategic Market Intelligence Program 2005

Anforderungen an Kontextwechselzeiten



Typische Anforderungen an den Kontextwechsel, Quelle: The Embedded Software Strategic Market Intelligence Program 2005



Echtzeitbetriebssysteme

OSEK



Hintergrund

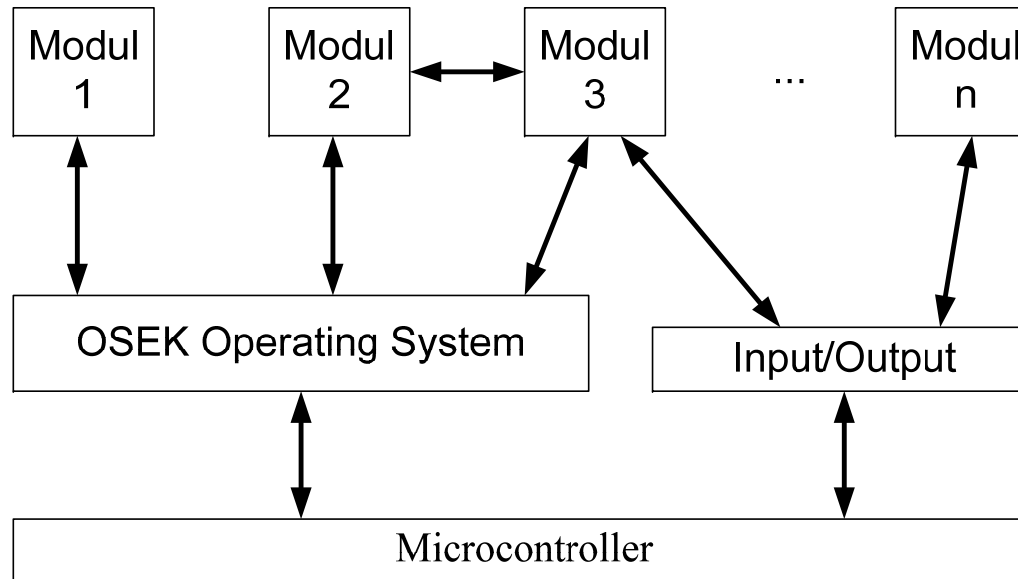
- Gemeinschaftsprojekt der deutschen Automobilindustrie (u.a. BMW, DaimlerChrysler, VW, Opel, Bosch, Siemens)
- OSEK: **O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**raftfahrzeug
- Ziel: Definition einer Standard-API für Echtzeitbetriebssysteme
- Standard ist frei verfügbar (<http://www.osek-vdx.org>), aber keine freien Implementierungen.
- Es existieren ebenso Ansätze für ein zeitgesteuertes Betriebssystem (OSEKTime), sowie eine fehlertolerante Kommunikationsschicht.



Anforderungen

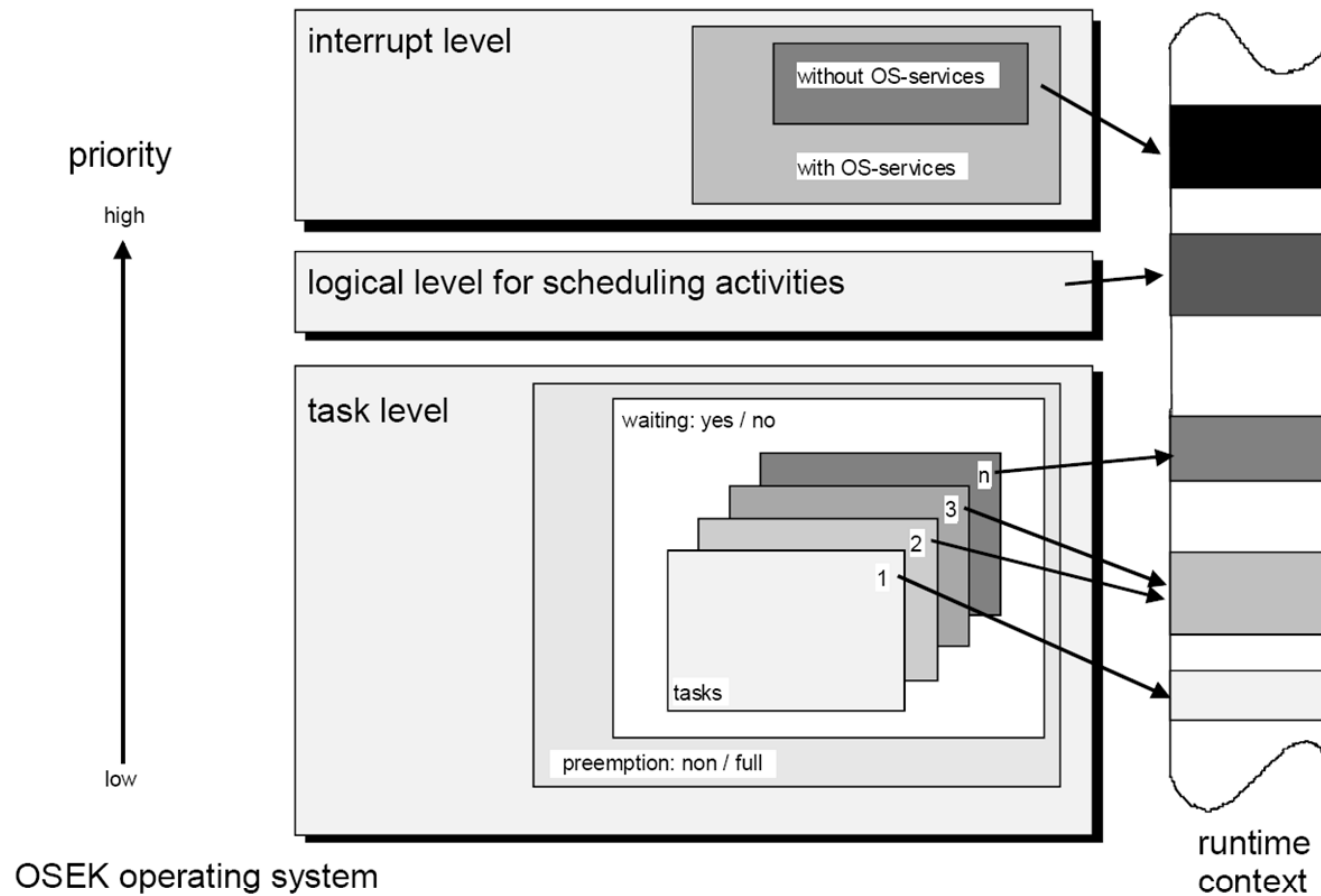
- Designrichtlinien bei der Entwicklung von OSEK:
 - harte Echtzeitanforderungen
 - hohe Sicherheitsanforderungen an Anwendungen
 - hohe Anforderungen an die Leistungsfähigkeit
 - typische: verteilte Systeme mit unterschiedlicher Hardware (v.a. Prozessoren)
- ⇒ typische Anforderungen von Echtzeitsystemen
- Weitere Ziele:
 - Skalierbarkeit
 - einfache Konfigurierbarkeit des Betriebssystems
 - Portabilität der Software
 - Statisch allokiertes Betriebssystem

OSEK Architektur



- Die Schnittstelle zwischen den einzelnen Anwendungsmodulen ist zur Erhöhung der Portierbarkeit standardisiert. Die Ein- und Ausgabe ist ausgelagert und wird nicht näher spezifiziert.

Ausführungsebenen in OSEK





Scheduling und Prozesse in OSEK

- Scheduling:
 - ausschließlich Scheduling mit statischen Prioritäten.
- Prozesse:
 - OSEK unterscheidet zwei verschiedene Arten von Prozessen:
 1. Basisprozesse
 2. Erweiterte Prozesse: haben die Möglichkeit über einen Aufruf der Betriebssystemfunktion `waitEvent()` auf externe asynchrone Ereignisse zu warten und reagieren.
 - Der Entwickler kann festlegen, ob ein Prozess unterbrechbar oder nicht unterbrechbar ist.
 - Es existieren somit vier Prozesszustände in OSEK: `running`, `ready`, `waiting`, `suspended`.



Betriebssystemklassen

- Der OSEK-Standard unterscheidet vier unterschiedliche Klassen von Betriebssystemen. Die Klassifizierung erfolgt dabei nach der Unterstützung:
 1. von mehrmaligen Prozessaktivierungen (einmalig oder mehrfach erlaubt)
 2. von Prozesstypen (nur Basisprozesse oder auch erweiterte Prozesse)
 3. mehreren Prozessen der selben Priorität
- Klassen:
 - BCC1: nur einmalig aktivierte Basisprozesse unterschiedlicher Priorität werden unterstützt.
 - BCC2: wie BCC1, allerdings Unterstützung von mehrmalig aufgerufenen Basisprozessen, sowie mehreren Basisprozessen gleicher Priorität.
 - ECC1: wie BCC1, allerdings auch Unterstützung von erweiterten Prozessen
 - ECC2: wie ECC1, allerdings Unterstützung von mehrmalig aufgerufenen Prozessen, sowie mehreren Prozessen gleicher Priorität.
- Die Implementierung unterscheidet sich vor allem in Bezug auf den Scheduler.



Unterbrechungsbehandlung

- In OSEK wird zwischen zwei Arten von Unterbrechungsbehandlern unterschieden:
 - ISR Kategorie 1: Der Behandler benutzt keine Betriebssystemfunktionen.
 - typischerweise die schnellsten und höchstpriorisierten Unterbrechungen.
 - Im Anschluss der Behandlung wird der unterbrochene Prozess fortgesetzt.
 - ISR Kategorie 2: Die Behandlungsroutine wird durch das Betriebssystem unterstützt, dadurch sind Aufrufe von Betriebssystemfunktionen erlaubt.
 - Falls ein Prozess unterbrochen wurde, wählt der Scheduler nach Beendigung der ISR den nächsten auszuführenden Prozess.



Prioritätsinversion

- Zur Vermeidung von Prioritätsinversion und Verklemmungen schreibt OSEK ein Immediate Priority Ceiling Protokoll vor:
 - Jeder Ressource wird eine Grenze (Maximum der Priorität der Prozesse, die die Ressource verwenden) zugewiesen.
 - Falls ein Prozess eine Ressource anfordert, wird die aktuelle Priorität des Prozesses auf die entsprechende Grenze angehoben.
 - Bei Freigabe fällt der Prozess auf die ursprüngliche Priorität zurück.

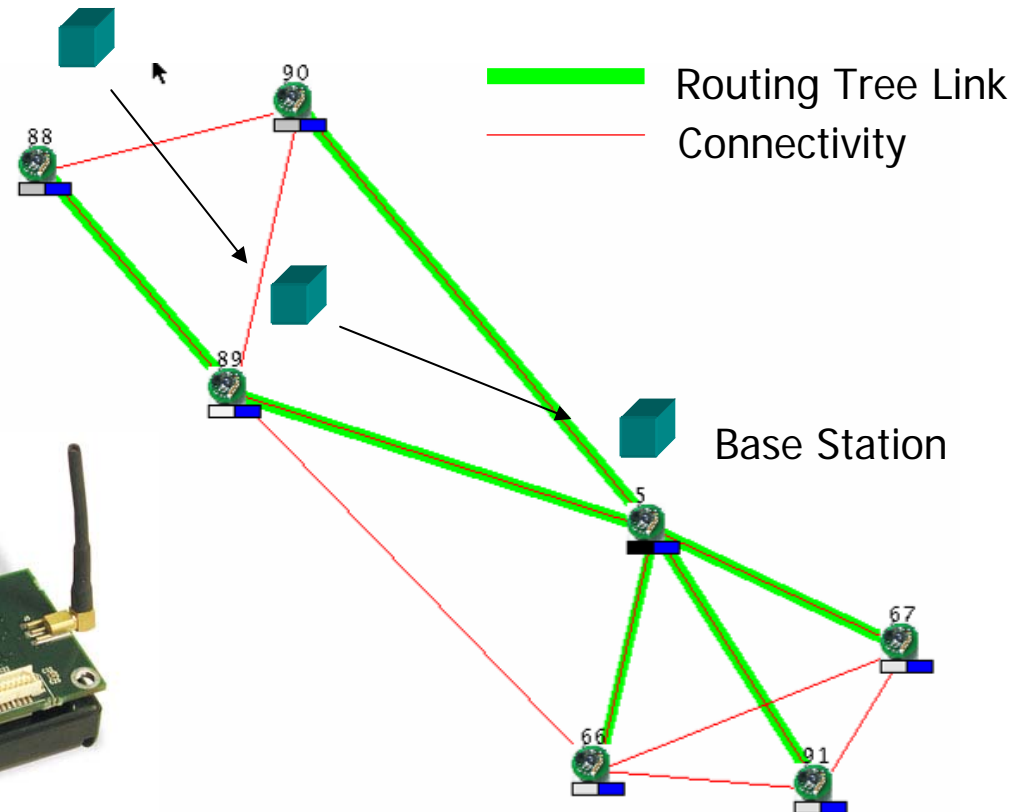


Echtzeitbetriebssysteme

TinyOS

Einsatzgebiet: AdHoc-Sensornetzwerke

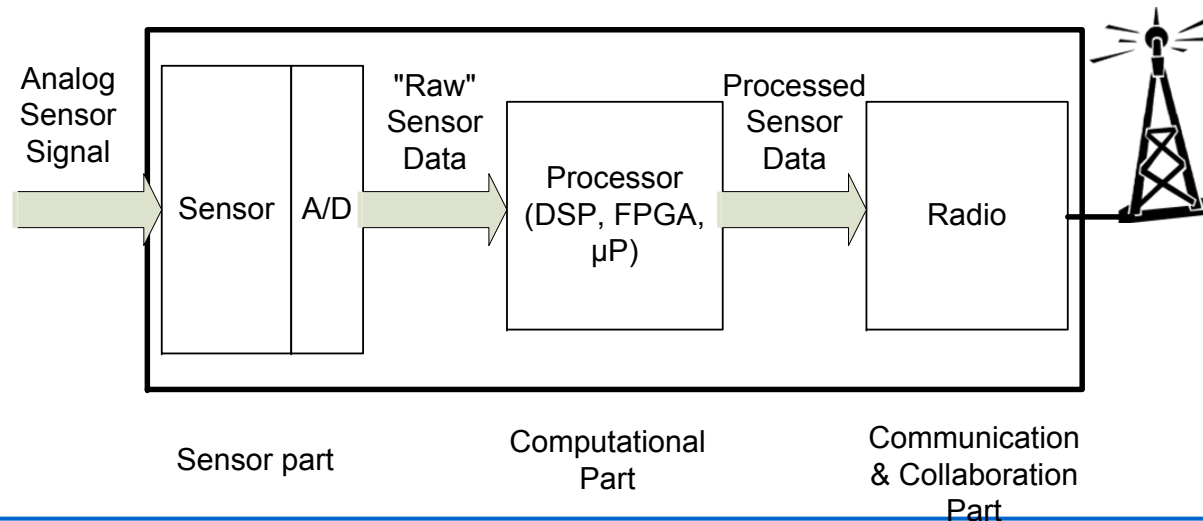
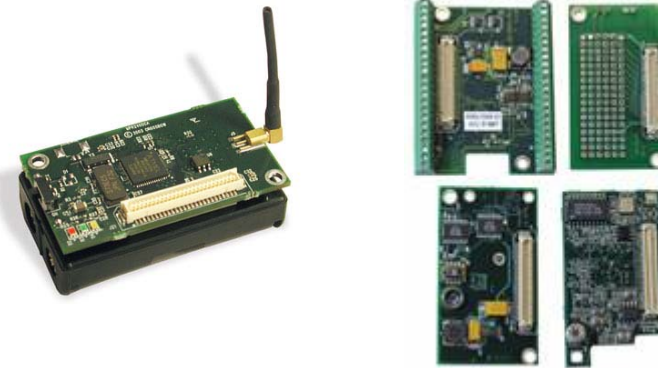
- Begriff Smart-Dust: Viele kleine Sensoren überwachen die Umgebung
- Ziele: robuste und flächendeckende Überwachung
- Probleme:
 - eingeschränkte Lebensdauer (Batterie)
 - eingeschränkter Speicherplatz
 - geringe Kommunikationsbandbreite
 - geringe Rechenleistung



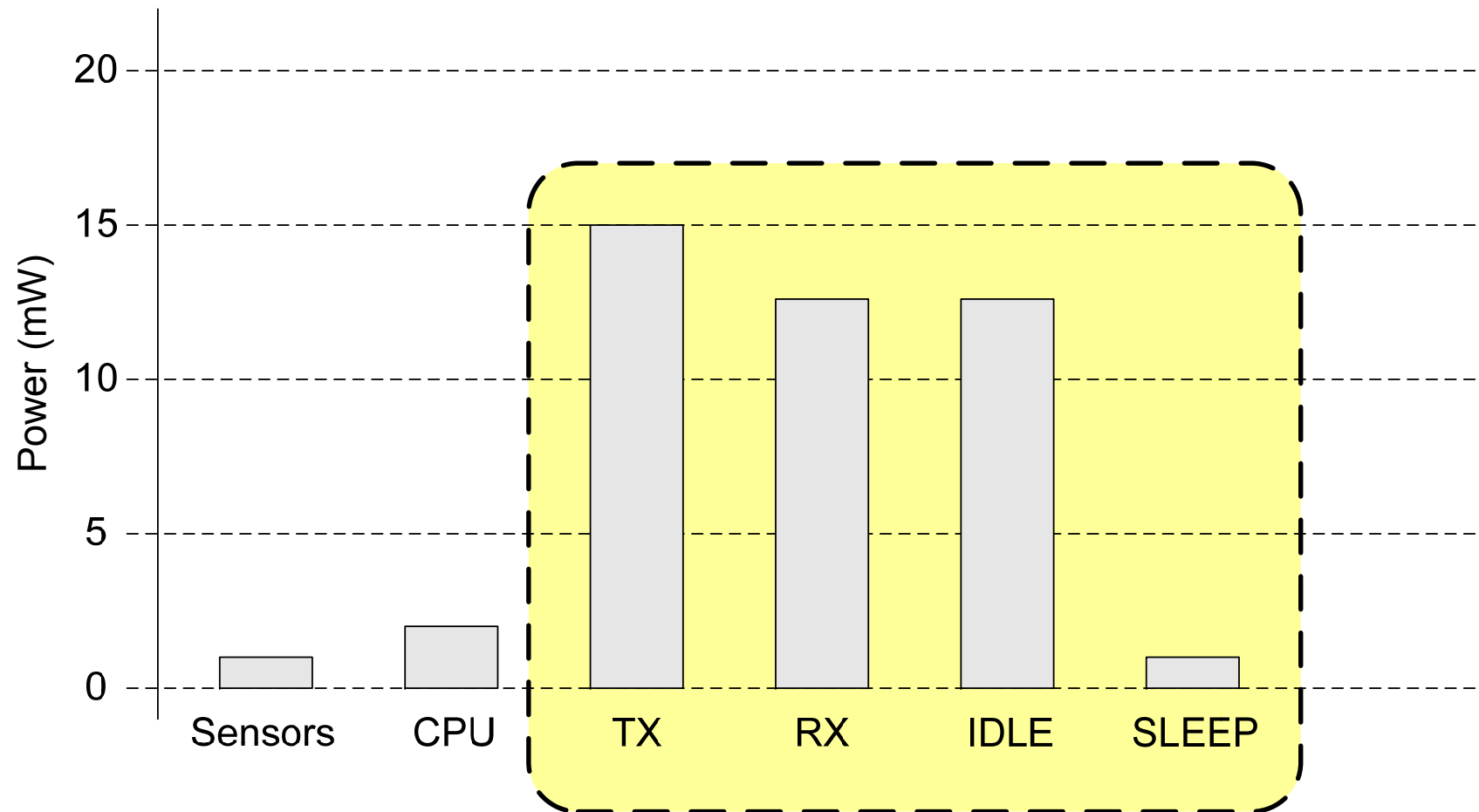
Quelle: <http://tinyos.millennium.berkeley.edu>

Hardware

- CPU: 4MHz, 8Bit, 512 Byte Ram
- Flash-Speicher: 128 kByte
- Funkmodul: 2,4 GHz, 250 kbps
- Diverse Sensormodule: z.B. Digital/Analog, Licht, Feuchtigkeit, Druck

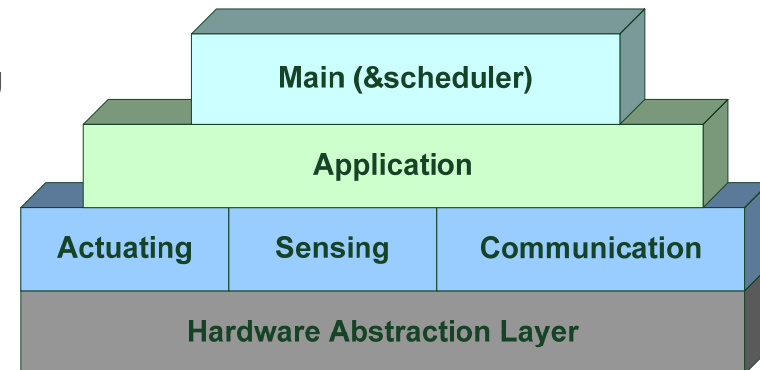


Stromverbrauch



TinyOS

- TinyOS ist kein wirkliches Betriebssystem im traditionellen Sinn, eher ein anwendungsspezifisches Betriebssystem
 - keine Trennung der Anwendung vom OS \Rightarrow Bei Änderung der Anwendung muss komplettes Betriebssystem neu geladen werden.
 - kein Kernel, keine Prozesse, keine Speicherverwaltung
 - Es existiert nur ein Stack (single shared stack)
- Ereignisbasiertes Ausführungsmodell
- Nebenläufigkeitskonzept:
 - Aufgaben können in unterschiedlichen Kontext ausgeführt werden:
 - Vordergrund: Unterbrechungsereignisse
 - Hintergrund: Tasks
 - Prozesse können durch Ereignisse, nicht jedoch durch andere Prozesse unterbrochen werden.
 - Scheduling für Tasks: Fifo
- Implementierung erfolgt in NesC (Erweiterung von C)
- Statische Speicherallokation





TinyOS - Architektur

