In Vorlesung betrachtete Echtzeitbetriebssysteme

- Der Markt der Echtzeitbetriebssysteme ist aufgrund der heterogenen Anforderungen sehr vielfältig
- Im Rahmen der Vorlesung betrachten wir verschiedene Vertreter, um typische Konzepte von Echtzeitbetriebssystemen darzustellen:
  - Anpassung an die Anforderungen der Anwendung: OSEK
  - Geringer Ressourcenverbrauch: TinyOS
  - Skalierbarkeit/Microkernelkonzept: QNX
  - Host-Target-Entwicklungsumgebung: VxWorks
  - Virtualisierung: PikeOS
- Zudem betrachten wir die Möglichkeiten Standardbetriebssysteme (Linux, Windows) für Echtzeitaufgaben einzusetzen

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.

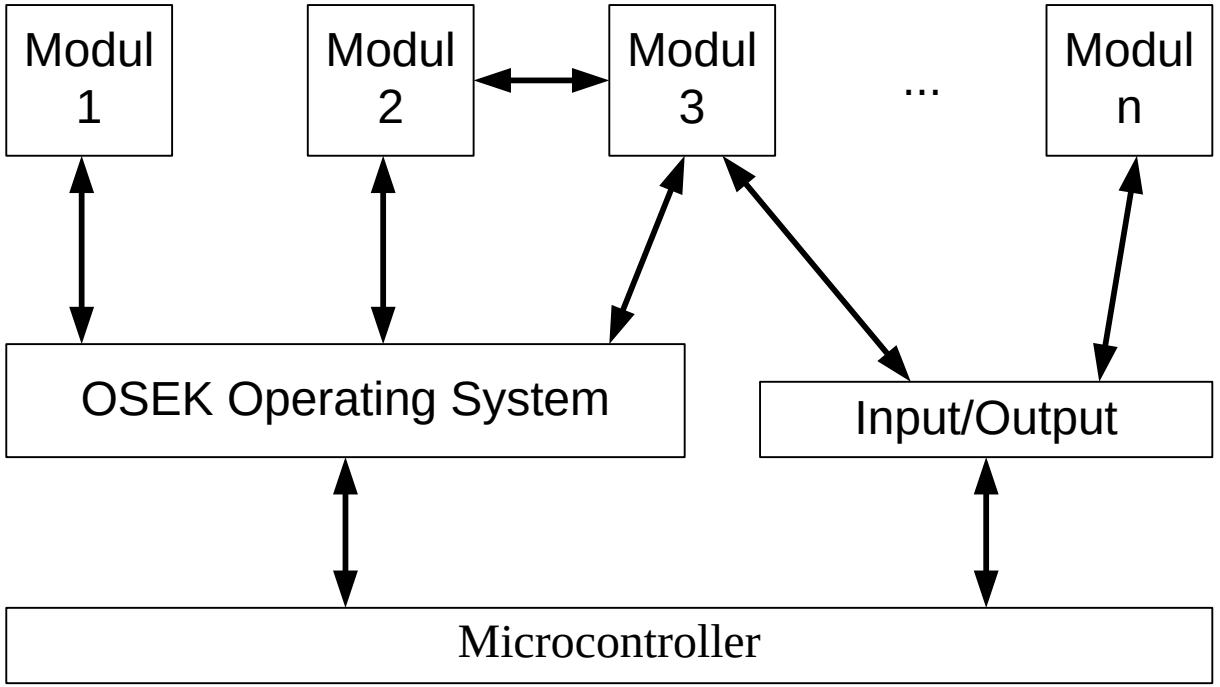
Anforderungen

- 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)

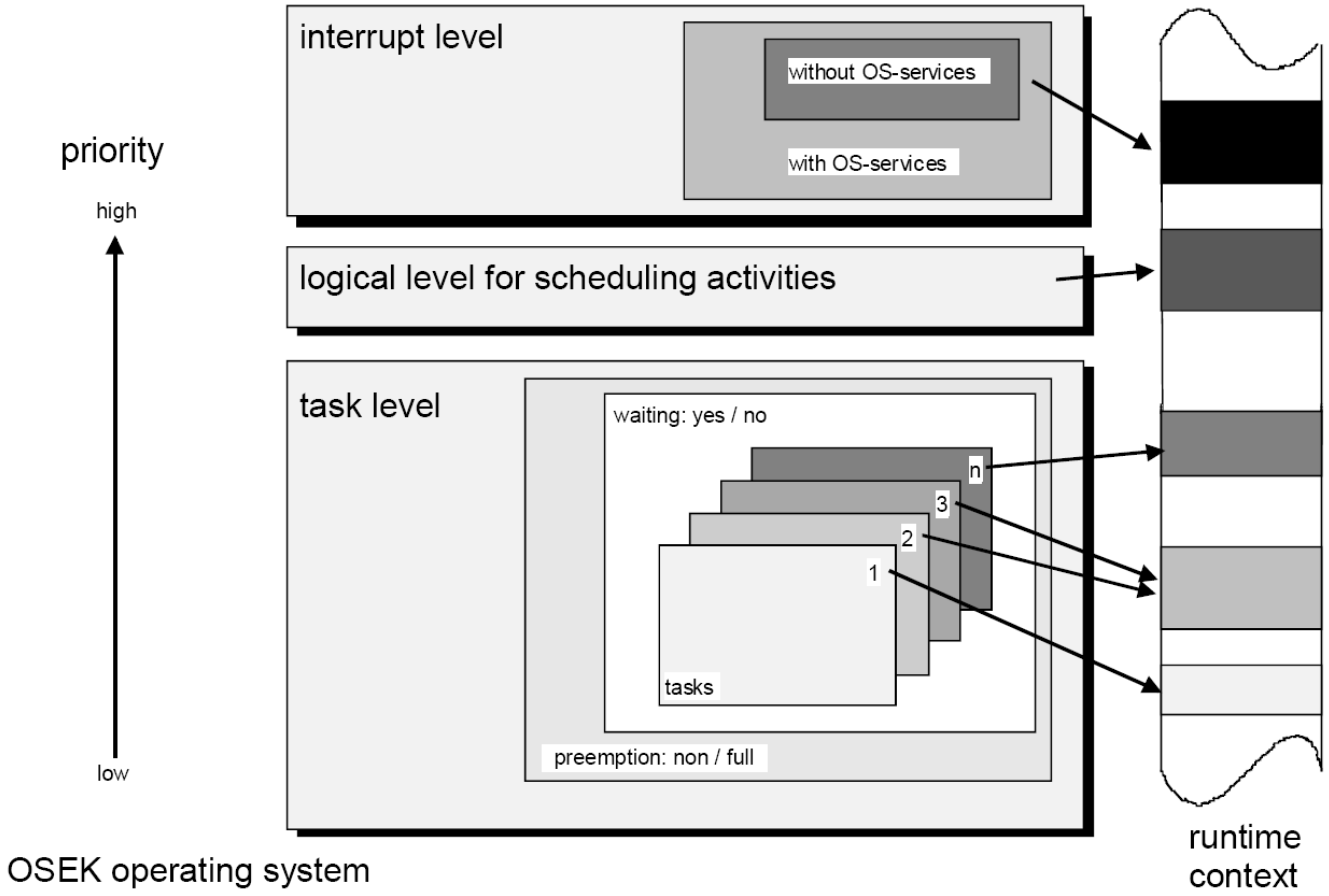
  → typische Anforderungen von Echtzeitsystemen
- Weitere Ziele:
  - Skalierbarkeit
  - einfache Konfigurierbarkeit des Betriebssystems
  - Portabilität der Software
  - Statisch allokiertes Betriebssystem

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 Betriebsystemen. 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

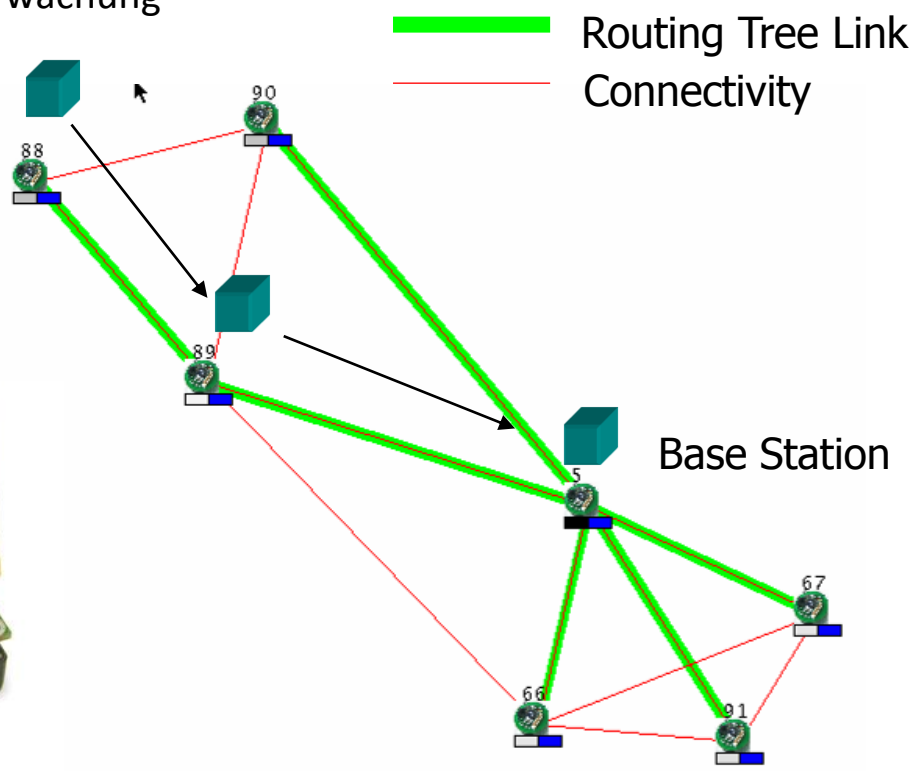
- Zur Vermeidung von Prioritätsinversion und Verklemmungen schreibt OSEK ein Immediate Priority Ceiling Protokoll vor:
  - Jeder Ressource wird eine Grenze (Maximum der Priorität der Prozesse, die die Ressource verwenden) zugewiesen.
  - Falls ein Prozess eine Ressource anfordert, wird die aktuelle Priorität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

Echtzeitbetriebssysteme

TinyOS

Einsatzgebiet: AdHoc-Sensornetzwerke

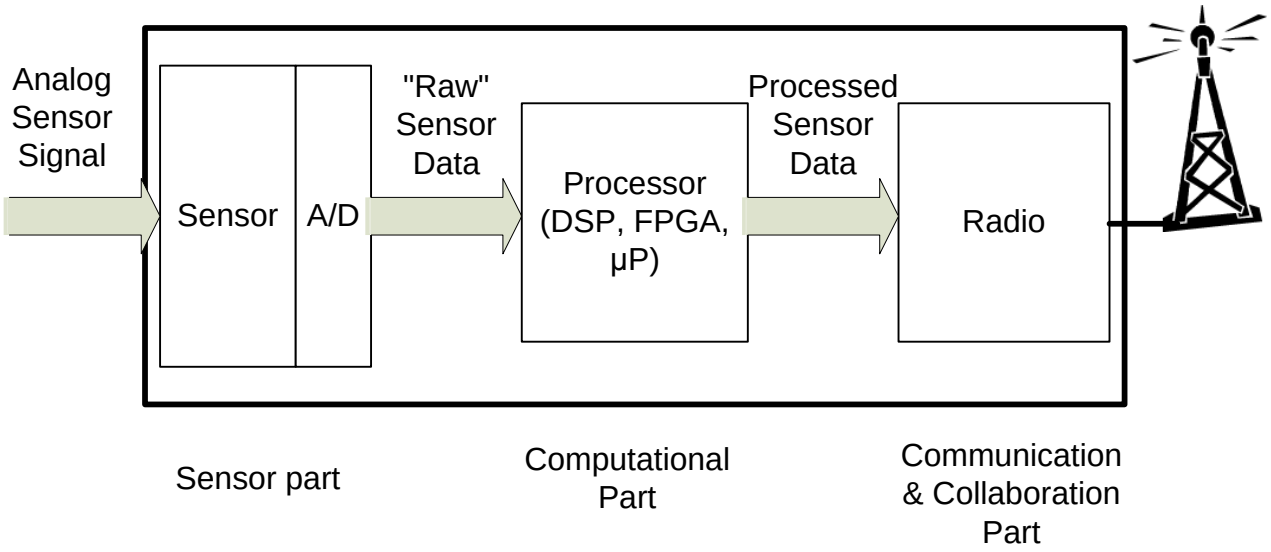
- Begriff Smart-Dust: Viele kleine Sensoren überwachen die Umgebung
- Ziele: robuste und flächendeckende Überwachung
- Probleme:
  - eingeschränkte Lebensdauer (Batterie)
  - eingeschränkter Speicherplatz
  - geringe Kommunikationsbandbreite
  - geringe Rechenleistung



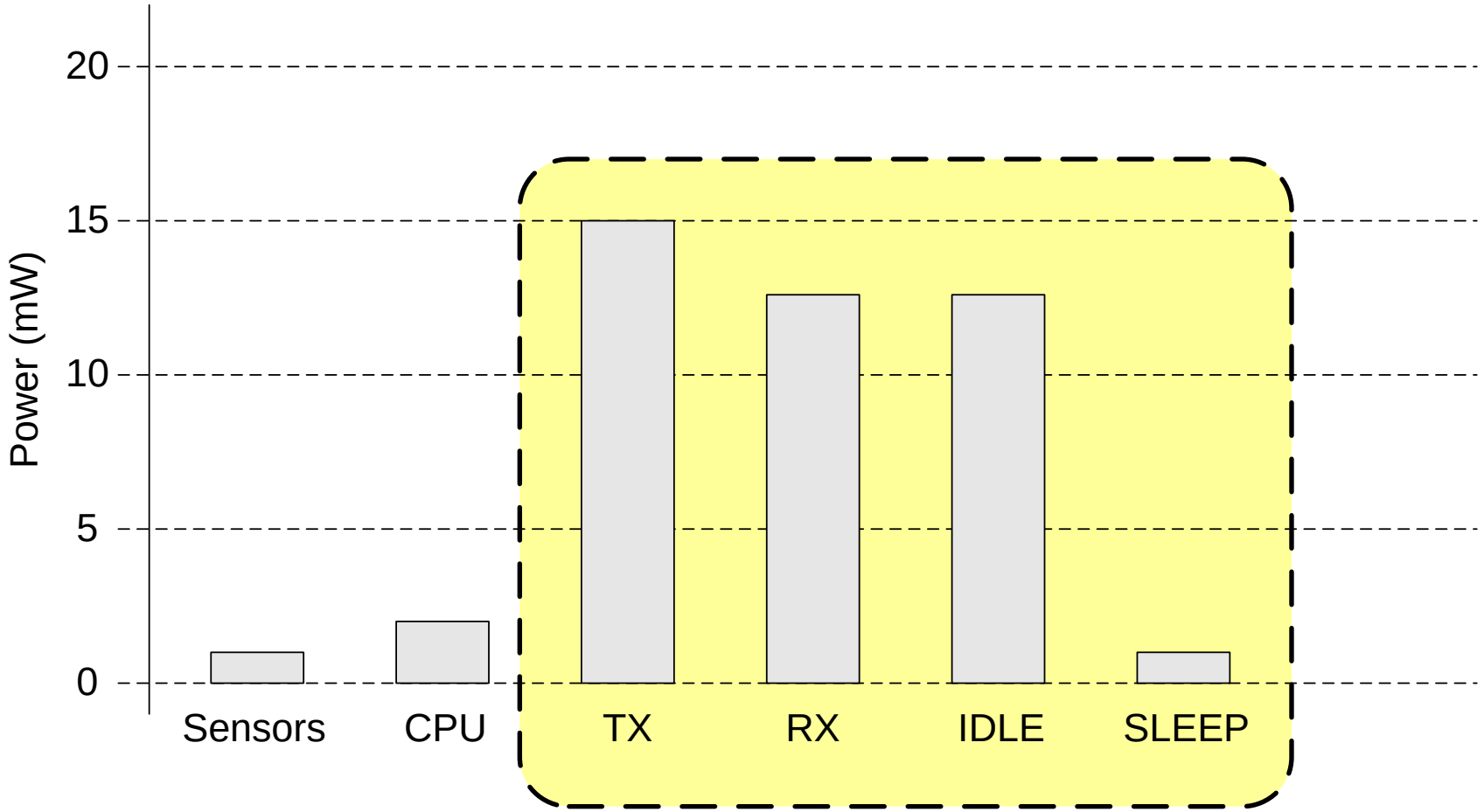
Quelle: http://tinyos.millennium.berkeley.edu

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

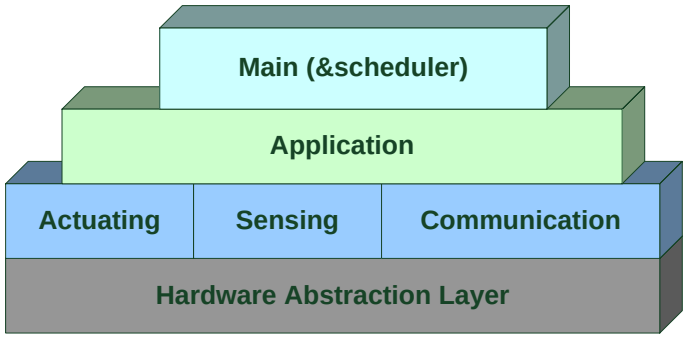


Stromverbrauch

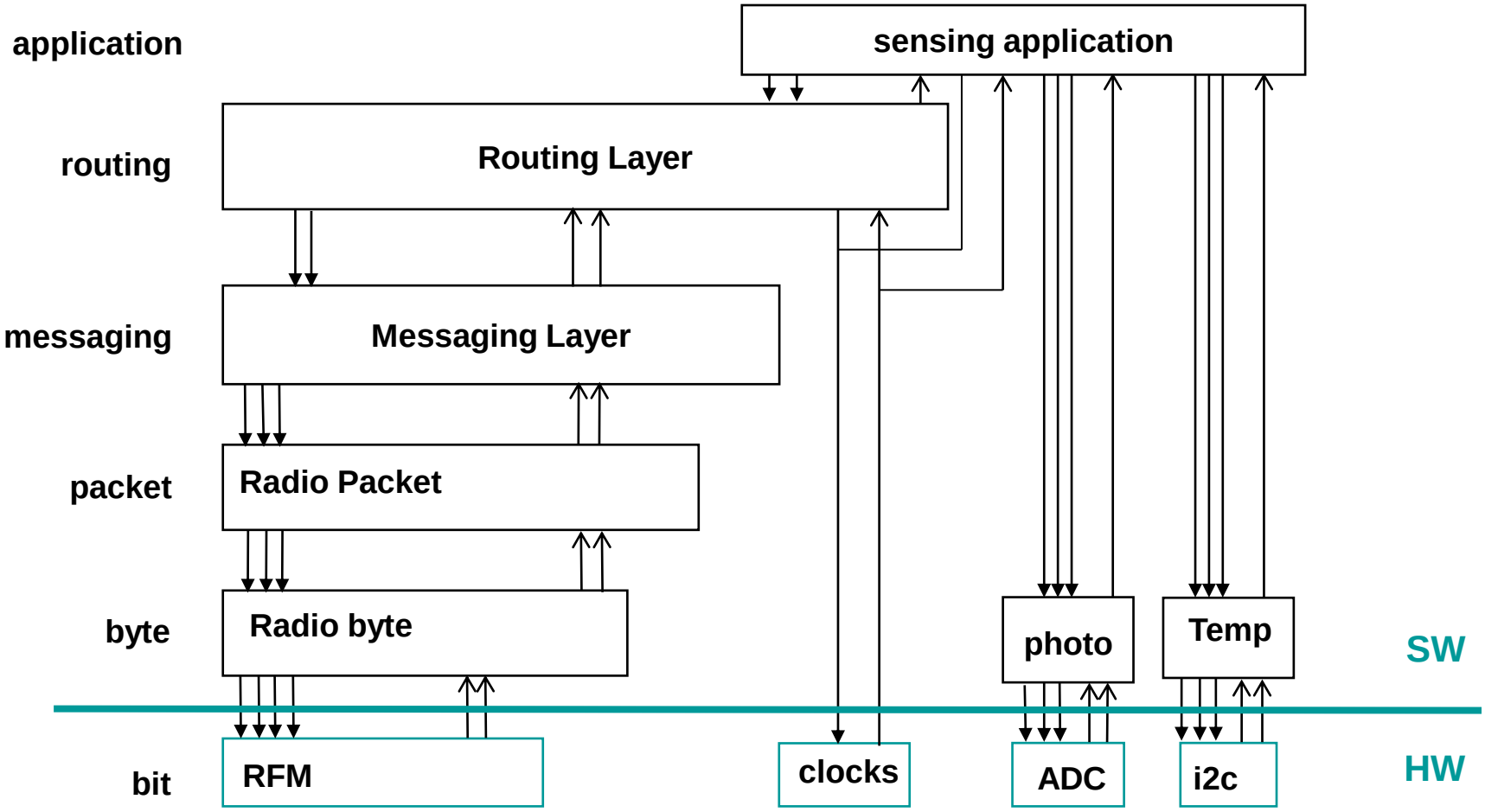


TinyOS

- TinyOS ist kein wirkliches Betriebssystem im traditionellen Sinn, eher ein anwendungs-spezifisches Betriebssystem
  - keine Trennung der Anwendung vom OS → 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



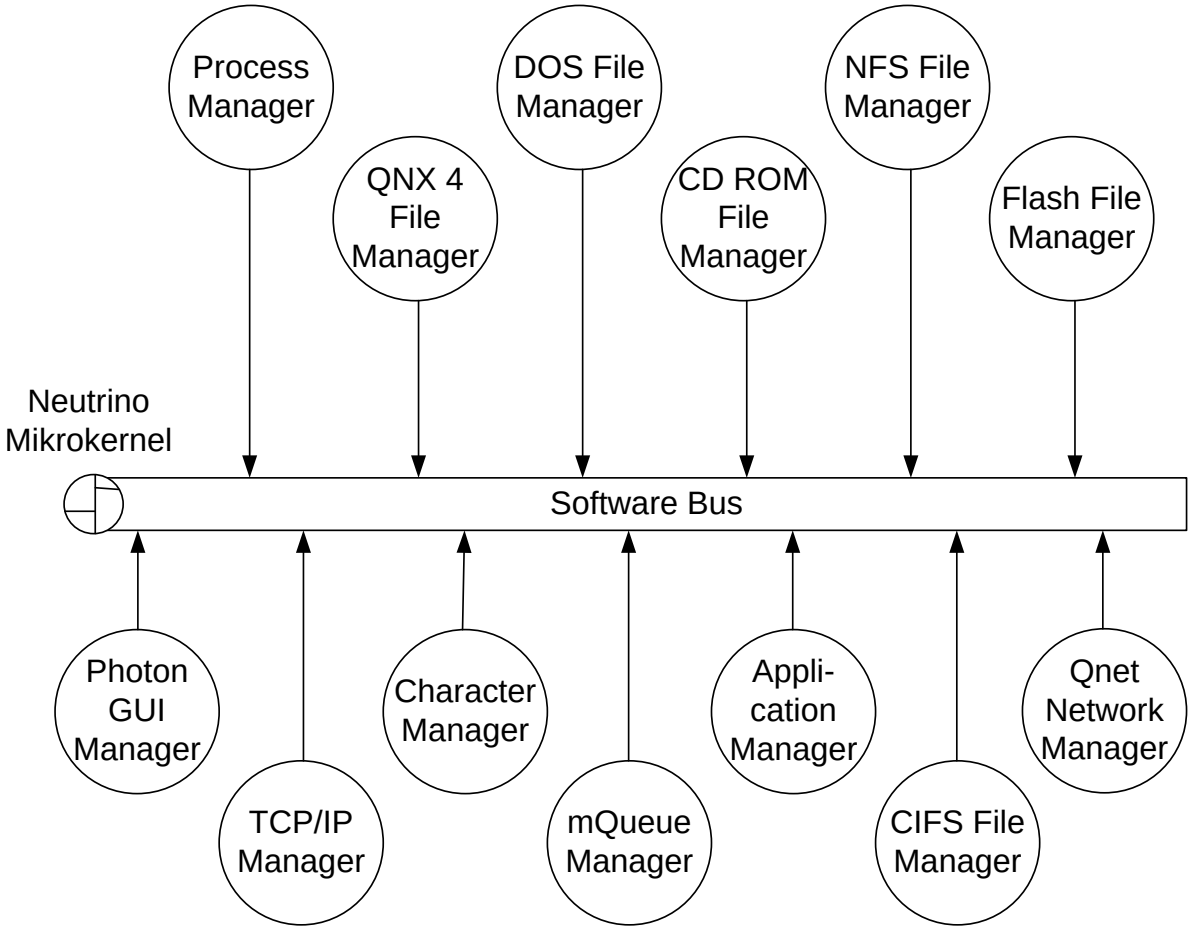
Echtzeitbetriebssysteme

QNX

Einführung

- Geschichte:
  - 1980 entwickeln Gordon Bell und Dan Dodge ein eigenes Echtzeitbetriebssystem mit Mikrokernel.
  - QNX orientiert sich nicht an Desktopsystemen und breitet sich sehr schnell auf dem Markt der eingebetteten Systeme aus.
  - Ende der 90er wird der Kernel noch einmal komplett umgeschrieben, um den POSIX-Standard zu erfüllen. → Ergebnis: QNX Neutrino.
- Besonderheiten von QNX
  - stark skalierbar, extrem kleiner Kernel (bei Version 4.24 ca.11kB)
  - Grundlegendes Konzept: Kommunikation erfolgt durch Nachrichten

QNX Architektur



Neutrino Microkernel

- Der Mikrokernel in QNX enthält nur die notwendigsten Elemente eines Betriebssystems:
  - Umsetzung der wichtigsten POSIX Elemente
    - POSIX Threads
    - POSIX Signale
    - POSIX Thread Synchronisation
    - POSIX Scheduling
    - POSIX Timer
  - Funktionalität für Nachrichten
- Eine ausführliche Beschreibung findet sich unter

  http://www.qnx.com/developers/docs/momentics621_docs/neutrino/sys_arch/kernel.html

Prozessmanager

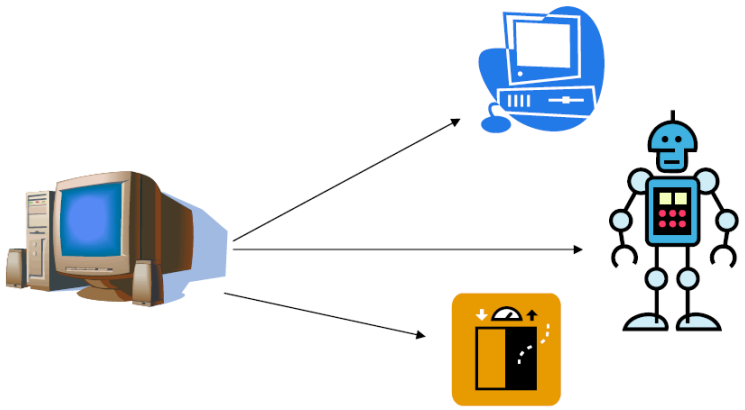
- Als wichtigster Prozess läuft in QNX der Prozessmanager.
- Die Aufgaben sind:
  - Prozessmanagement:
    - Erzeugen und Löschen von Prozessen
    - Verwaltung von Prozesseigenschaften
  - Speichermanagement:
    - Bereitstellung von Speicherschutzmechanismen,
    - von gemeinsamen Bibliotheken
    - und POSIX Primitiven zu Shared Memory
  - Pfadnamenmanagement
- Zur Kommunikation zwischen und zur Synchronisation von Prozessen bietet QNX Funktionalitäten zum Nachrichtenaustausch an.

Echtzeitbetriebssysteme

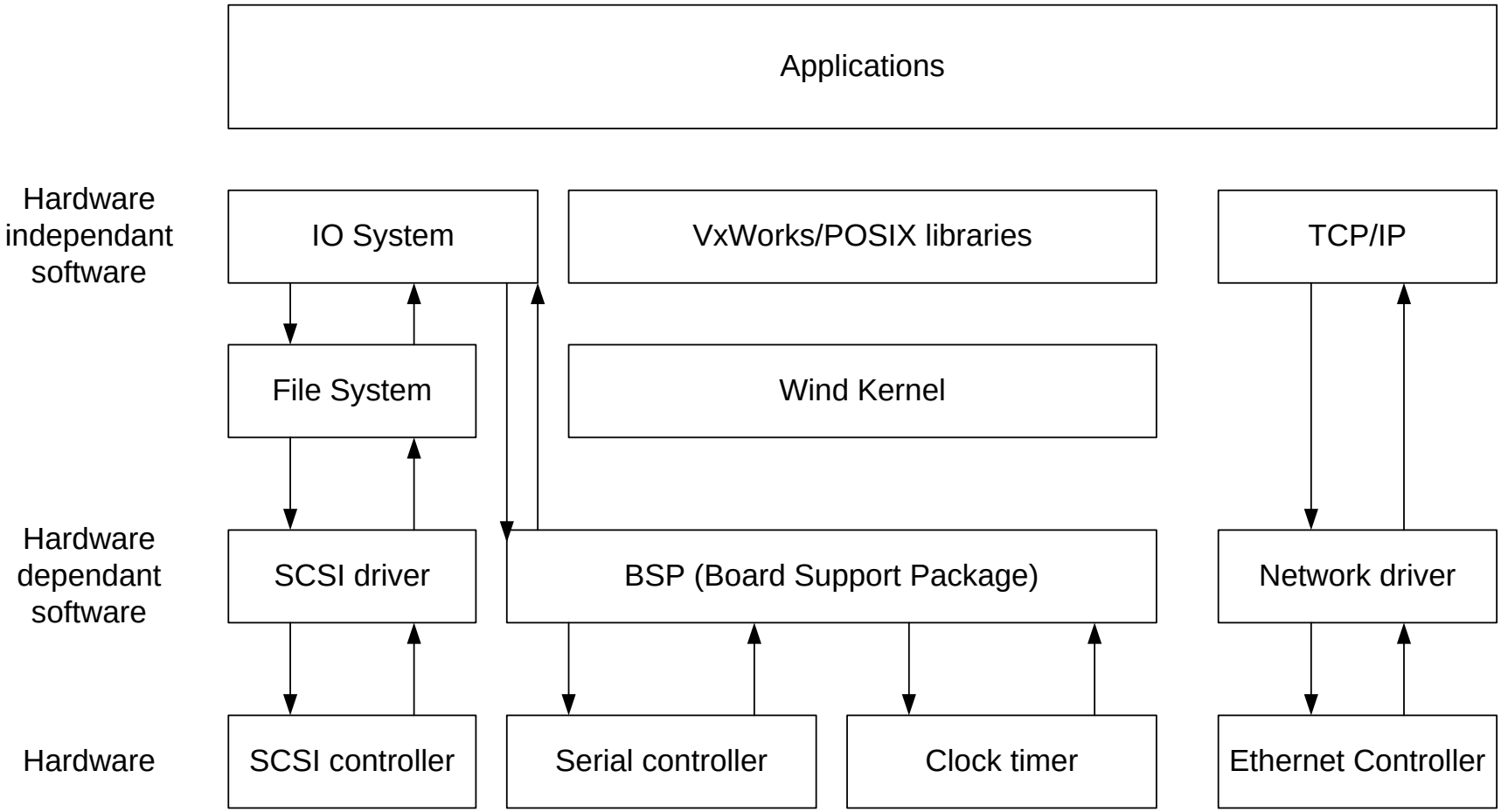
VxWorks

Eigenschaften

- Host-Target-Entwicklungssystem
- Eigene Entwicklungsumgebung Workbench mit Simulationsumgebung und integriertem Debugger basierend auf Eclipse
- Zielplattformen der Workbench 2.0: VxWorks, Linux Kernel 2.4/2.6
- Auf der Targetshell wird auch ein Interpreter ausgeführt → C-Code kann direkt in die Shell eingegeben werden
- Kernel kann angepasst werden, allerding muss der Kernel dazu neu kompiliert werden
- Marktführer im Bereich der Echtzeitbetriebssysteme



Architektur



Prozessmanagement

- **Schedulingverfahren**: Es werden nur die beiden Verfahren FIFO und RoundRobin angeboten. Ein Verfahren für periodische Prozesse ist nicht verfügbar.
- **Prioritäten**: Die Prioritäten reichen von 0 (höchste Priorität) bis 255.
- **Prozessanzahl**: Die Anzahl der Prozesse ist nicht beschränkt (aber natürlich abhängig vom Speicherplatz)
- **API**: VxWorks bietet zum Management von Prozessen eigene Funktionen, sowie POSIX-Funktionen an.

Interprozesskommunikation und Speichermanagement

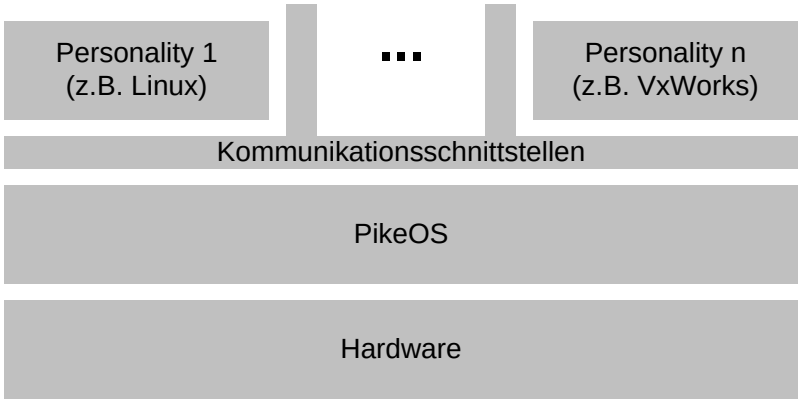
- Zur Interprozesskommunikation werden folgende Konzepte unterstützt:
  - Semaphor
  - Mutex (mit Prioritätsvererbung)
  - Nachrichtenwartschlangen
  - Signale
- Seit Version 6.0 wird zudem Speichermanagement angeboten:
  - Der Entwickler kann Benutzerprozesse mit eigenem Speicherraum entwickeln.
  - Bisher: nur Threads im Kernel möglich.

Echtzeitbetriebssysteme

PikeOS

PikeOS: Betriebssystem mit Paravirtualisierung

- Idee: Virtualisierung der Hardware – jede Partition (Personality) verhält sich als hätte sie eine eigene CPU zur Verfügung
- Mehrere Betriebssysteme können auf der gleichen CPU nebenläufig ausgeführt werden.
- Die Speicherbereiche, sowie CPU-Zeiten der einzelnen Partitionen werden statisch während der Implementierung festgelegt.
- Durch die Partitionierung ergeben sich diverse Vorteile:
  - Bei einer Zertifizierung muss nur der sicherheitskritische Teil des Gesamtsystems zertifiziert werden.
  - Reduzierung der Steuergeräte durch Zusammenführung der Funktionalitäten mehrerer Steuergeräte
  - Echtzeitkomponenten können einfacher von nicht-kritischen Komponenten getrennt werden – Nachweis der Fristeneinhaltung wird einfacher



Architektur

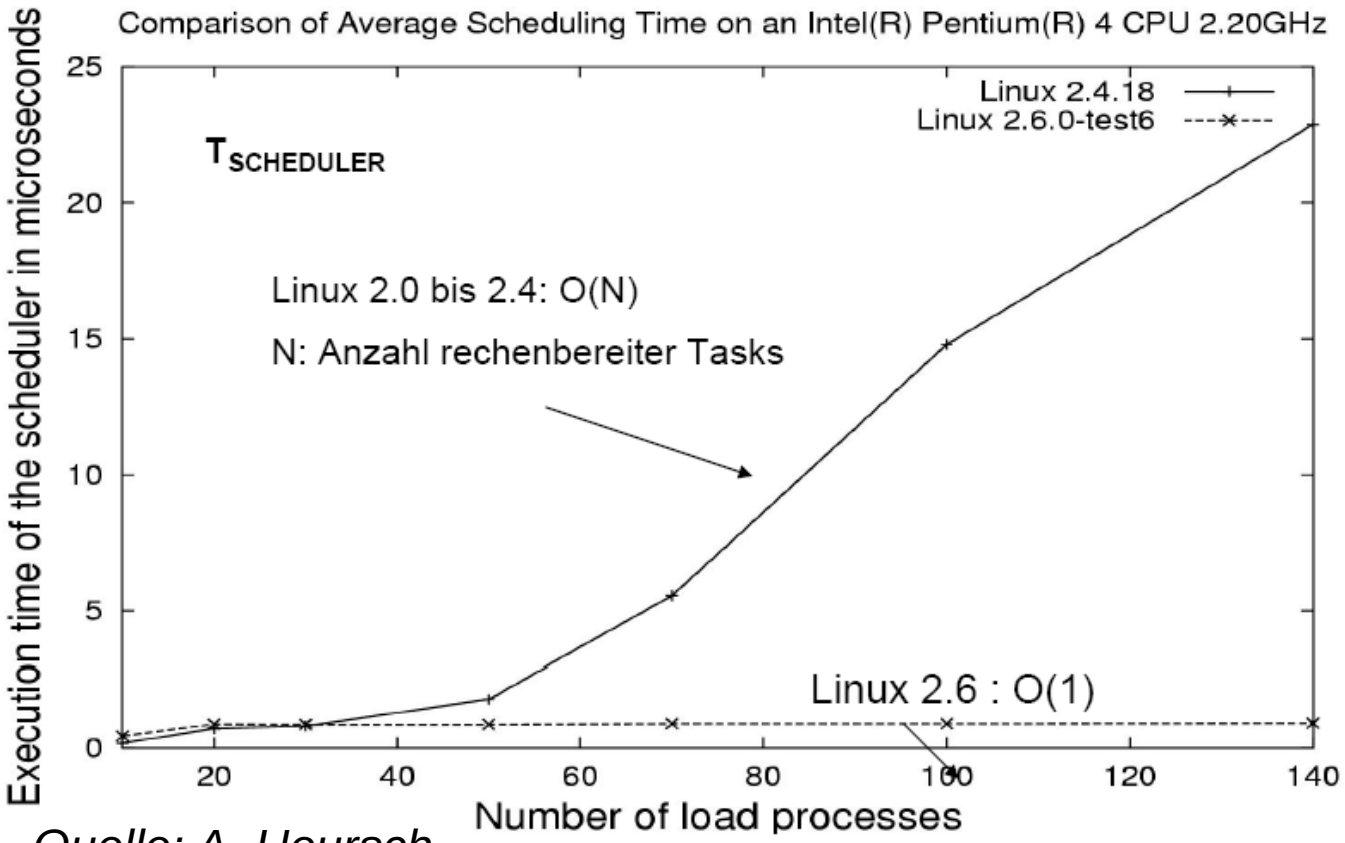
Echtzeitbetriebssysteme

Linux Kernel 2.6

Bestandsaufnahme

- Für die Verwendung von Linux Kernel 2.6 in Echtzeitsystemen spricht:
  - die Existenz eines echtzeitfähigen Schedulingverfahrens (prioritätenbasiertes Scheduling mit FIFO oder RoundRobin bei Prozessen gleicher Priorität)
  - die auf 1 ms herabgesetzte Zeitauflösung der Uhr (von 10ms in Kernel 2.4)
- Gegen die Verwendung spricht:
  - die Ununterbrechbarkeit des Kernels.

Vergleich Schedulerlaufzeiten Kernel 2.4/2.6



Quelle: A. Heursch

Unterbrechbarkeit des Kernels

- Im Kernel ist der **Preemptible Kernel Patch** als Konfigurationsoption enthalten → Erlaubt die Unterbrechung des Kernels.
- **Problem**: Existenz einer Reihe von kritischen Bereichen, die zu langen Verzögerungszeiten führen.
- **Low Latency Patches** helfen bei der Optimierung, aber harte Echtzeitanforderungen können nicht erfüllt werden.
- Weitere Ansätze: z.B. Verwendung von binären Semaphoren (Mutex) anstelle von generellen Unterbrechungssperren, Verhinderung von Prioritätsinversion durch geeignete Patches, siehe Paper von A. Heursch

Speichermanagement

- Linux unterstützt Virtual Memory
- Die Verwendung von Virtual Memory führt zu zufälligen und nicht vorhersagbaren Verzögerung, falls sich eine benötigte Seite nicht im Hauptspeicher befindet.
  → Die Verwendung von Virtual Memory in Echtzeitanwendungen ist nicht sinnvoll.
- Vorgehen: Zur Vermeidung bietet Linux die Funktionen `mlock()` und `mlockall()` zum **Pinning** an.
- Pinning bezeichnet die Verhinderung des Auslagerns eines bestimmten Speicherbereichs oder des kompletten Speichers eines Prozesses.

Uhrenauflösung

- Die in Linux Kernel 2.6 vorgesehene Uhrenauflösung von 1ms ist häufig nicht ausreichend.
- Problemlösung: Verwendung des **High Resolution Timer Patch (hrtimers)**
  - Durch Verwendung des Patches kann die Auflösung verbessert werden.
  - Der Patch erlaubt z.B. die Erzeugung einer Unterbrechung in 3,5 Mikrosekunden von jetzt an.
  - Einschränkung: Zeitliche Angabe muss schon vorab bekannt sein → keine Zeitmessung möglich
  - Gründe für die hrtimers-Lösung findet man unter: http://www.kernel.org/git/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=Documentation/hrtimers.txt

Echtzeitbetriebssysteme

RTLinux/RTAI

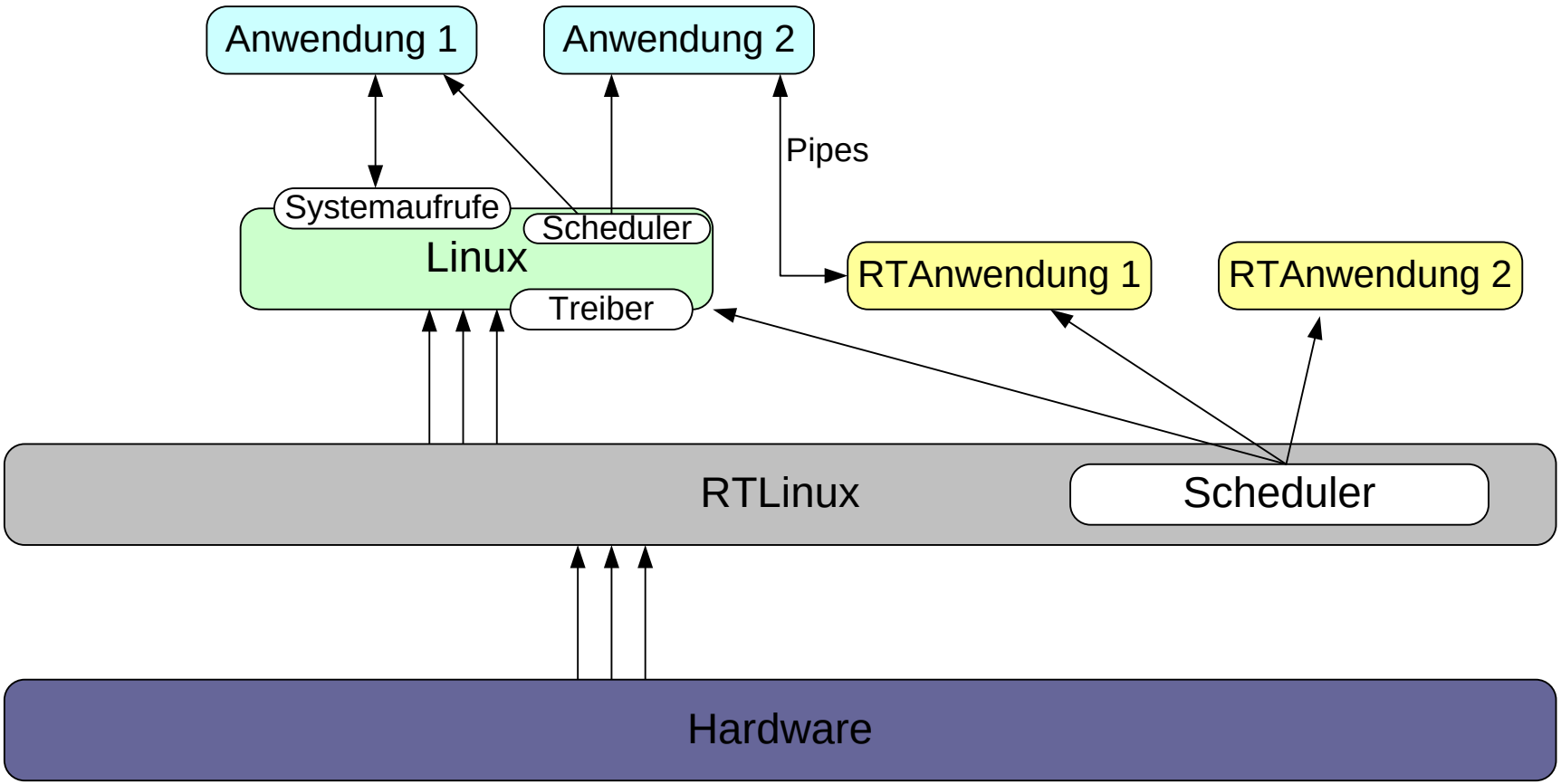
Motivation

- Aus diversen Gründen ist die Verwendung von Linux in Echtzeitsystemen erstrebenswert:
  - Linux ist weitverbreitet
  - Treiber sind sehr schnell verfügbar
  - Es existieren viele Entwicklungswerkzeuge → die Entwickler müssen nicht für ein neues System geschult werden.
  - Häufig müssen nur geringe Teile des Codes echtzeitfähig ausgeführt werden.
- **Probleme**:
  - grobgranulare Synchronisation
  - trotz Patches oft zu lange Latenzzeiten
  - Hochpriorisierte Prozesse können durch andere Prozesse mit niedrigerer Priorität blockiert werden, Grund: Hardwareoptimierungsstrategien (z.B. Speichermanagement)
- **Ansatz**: Modifikation von Linux, so dass auch harte Echtzeitanforderungen erfüllt werden.

Ansatz

- Anstelle von Patches wird eine neue Schicht zwischen Hardware und Linux-Kernel eingefügt:
  - Volle Kontrolle der Schicht über Unterbrechungen
  - Virtualisierung von Unterbrechungen (Barabanov, Yodaiken, 1996): Unterbrechungen werden in Nachrichten umgewandelt, die zielgerichtet zugestellt werden.
  - Virtualisierung der Uhr
  - Anbieten von Funktionen zum virtuellen Einschalten und Ausschalten von Unterbrechungen
  - Das Linux-System wird als Prozess mit niedrigster Priorität ausgeführt.

RTLinux Architektur



Unterschiede RTAI/RTLinux

- RTLinux verändert Linux-Kernel-Methoden für den Echtzeiteingriff → Kernel-Versions-Änderungen haben große Auswirkungen.
- RTAI fügt Hardware Abstraction Layer (HAL) zwischen Hardware und Kernel ein. Hierzu sind nur ca. 20 Zeilen Code am Originalkern zu ändern. HAL selbst umfasst kaum mehr als 50 Zeilen → Transparenz.
- RTAI ist frei, RTLinux in freier (Privat, Ausbildung) und kommerzieller Version.
- Beide Ansätze verwenden ladbare Kernel Module für Echtzeitprozesse.
- RTAI (mit Variante LXRT) erlaubt auch die Ausführung von echtzeitkritischen Prozessen im User-Space, Vorteil ist beispielsweise der Speicherschutz

Echtzeitbetriebssysteme

Windows CE & Windows Embedded

Eigenschaften

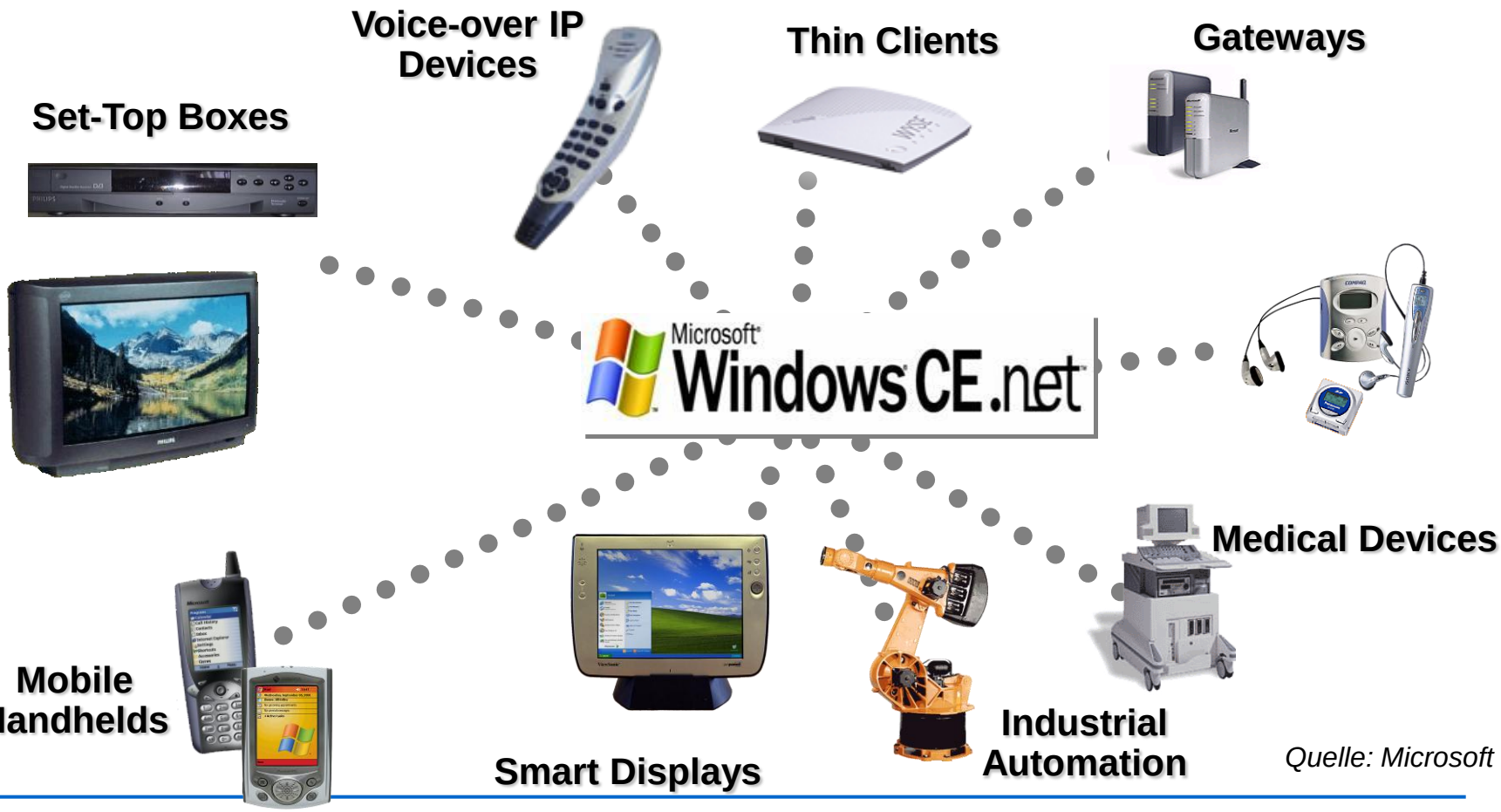
- Windows CE
  - 32-bit, Echtzeitbetriebssystem
  - Unterstützung von Multitasking
  - Stark modularer Aufbau
  - Skalierbar entsprechend der gewünschten Funktionalität
- Windows Embedded
  - „Skalierbares Windows XP“
  - Komponenten von XP können entfernt werden um den benötigten Speicherplatz zu minimieren

Windows CE und Embedded im Vergleich

x86 processors	Processor Support	Multiple processors / power management
Full Win32 API compatibility	Win32 API Compatibility	Requires additional effort
Basic images from 8MB (“Hello World”)	Footprint	Basic images from 350 KB
With 3rd party extensions	Real-time	Native

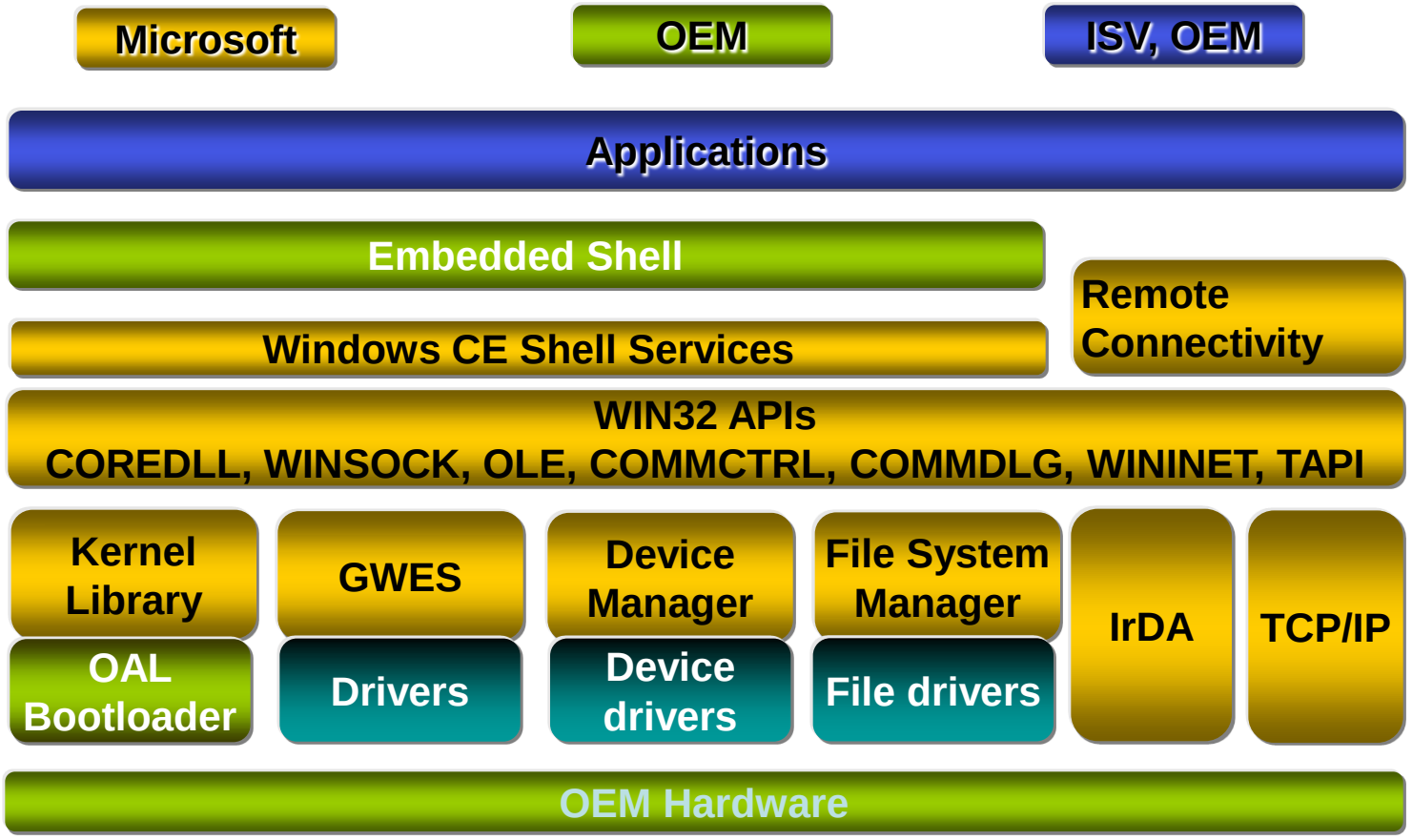
Quelle: Microsoft

Einsatzbereiche



Quelle: Microsoft

Windows CE Architektur



Quelle: Microsoft

Funktionen des Betriebssystemkerns

- Kernel, Speicherverwaltung
  - Shared heap
  - Unterstützung von Watchdogs
  - 64 Systeminterrupts
- Geräteunterstützung
  - Unterstützung diverser Massenspeicher, z.B. USB, Flash,..
- Browser
- Multimedia
  - Diverse Graphiktreiber
  - umfassende Codecunterstützung
- Kryptographie-Funktionen

Echtzeitunterstützung

- Unterstützung verschachtelter Interrupts
- 256 Prioritätslevel
- Thread quantum level control
- Speicherschutz (Pinning) zur Umgehung von Virtual Memory
- Eingebaute Leistungsüberwachungswerkzeuge
- Niedrige ISR/IST Latenz
  - ISR/IST Latenz von 2.8/26.4 Mikrosekunden auf Intel 100MHz Board

Echtzeitbetriebssysteme

Zusammenfassung

Zusammenfassung

- Es gibt kein typisches Echtzeitbetriebssystem da je nach Einsatzbereich die Anforderungen sehr unterschiedlich sind.
- Der minimale Speicherbedarf reicht von wenigen Kilobyte (TinyOS, QNX) bis hin zu mehreren Megabyte (Windows CE / XP Embedded).
- Die Betriebssysteme sind typischerweise skalierbar. Zur Änderung des Leistungsumfangs von Betriebssystemen muss das System entweder neu kompiliert werden (VxWorks) oder neue Prozesse müssen nachgeladen werden (QNX).
- Die Echtzeitfähigkeit von Standardbetriebssysteme kann durch Erweiterungen erreicht werden (RTLinux/RTAI).
- Die Schedulingverfahren und die IPC-Mechanismen orientieren sich stark an den in POSIX vorgeschlagenen Standards.
- Das Problem der Prioritätsinversion wird zumeist durch Prioritätsvererbung gelöst.

Erfolgskontrolle: Was Sie aus dem Kapitel mitgenommen haben sollten?

- Kenntniss der besonderen Anforderungen von Echtzeitsystemen an Betriebssysteme
- Wesentliche Konzepte der in der Vorlesung besprochenen Betriebssysteme
- Zuordnung von Anwendungsklassen zu den verschiedenen Betriebssystemen: „wann wurde man welches Betriebssystem einsetzen?“
- Möglichkeiten zum Einsatz von Standardbetriebssystemen

Klausurfragen

- Wiederholungsklausur WS 2006/2007 (5 Punkte = 5 min)
  - Erläutern Sie die Unterschiede zwischen Betriebssystemen mit kooperativem Scheduling, mit präemptiven Scheduling und präemptiblen Betriebssystemen.
  - Lösung:
    - Beim kooperativen Scheduling bekommt ein Prozess den Prozessor bis zu dem Zeitpunkt an dem dieser freiwillig freigegeben wird, der Prozessor kann nicht entzogen werden.
    - Beim präemptiven Scheduling kann der Prozessor dem Prozess entzogen werden, solange dieser nicht im Kernelbereich ausgeführt wird.
    - Beim präemptiblen Scheduling kann der Prozessor einem Prozess auch bei Ausführung im Kernelbereich entzogen werden.
- Klausur WS 2007/2008 (4 Punkte = 4 min)
  - Erläutern Sie kurz (jeweils 1-2 Sätze) die Hauptkonzepte von TinyOS, QNX und PikeOS.
  - Lösung: TinyOS ist für den Einsatz in ressourcenbeschränkten Systemen konzipiert und ist eher als Middleware anzusehen, die an die Anwendung angepasst werden kann. QNX hat einen sehr kleinen, modularen Kernel und benutzt zur Interaktion der Komponenten Nachrichten. PikeOS bietet eine Virtualisierung der Hardware an und erlaubt die Ausführung unterschiedlicherBetriebssysteme mit klaren Speicher- und Zeitzuteilungen auf einer CPU.

Klausurfragen - Klausur WS 2006/2007 (10 Punkte = 10 min)

Gegeben seien folgende fünf Echtzeitbetriebssysteme:

1. TinyOS
2. OsekTime
3. QNX
4. VxWorks
5. WindowsCE

und folgende fünf Anwendungen:

a) Sicherheitsüberwachung für Robotersteuerung: auf Basis eines Controllers mit geringen Rechen- und Speicherkapazitäten wird eine Anwendung zur Überwachung der Robotersteuerung implementiert. Dringt eine Person in den Arbeitsbereich des Roboters ein, so wird dieses Ereignis durch Lichtschranken detektiert und der Roboter unverzüglich gestoppt.

b) ZebraNet: Zur Erforschung der Wanderwege von Zebras, werden kleine Funkmodule ausgestattet mit GPS-Sensoren zur Lokalisation und Solarzellen zur Stromversorgung an Zebras angebracht. Nähern sich zwei Zebras, so tauschen die Funkmodule die gesammelten Daten aus.

c) Zugsteuerung: Zur Steuerung und Überwachung eines Zuges werden in einem verteilten System verschiedene periodische Regelungsfunktionen ausgeführt. Diese Funktionen werden dabei als Komponenten von unterschiedlichen Entwicklergruppen zur Verfügung gestellt. Insbesondere eine reibungslose Integration dieser Komponenten steht im Vordergrund.

d) Fabrikanlagensteuerung: In einer Chemiefabrik wird die Produktion von leistungsfähigen Feldrechnern gesteuert. Diese Rechner sind mit einer Vielzahl von Sensorik verbunden. Auf kritische Ereignisse muss schnell reagiert werden.

e) PDA: Auf einem Handheld werden unterschiedlichste Anwendungen ausgeführt: unter anderem Routenplaner, Browser, Musik-/Videospieler und Programme zur Terminverwaltung.

Ordnen Sie jedem der fünf Anwendungsgebiete ein Echtzeitbetriebssystem zu und begründen Sie Ihre Zuordnung knapp mit einem Satz.

Klausurfragen - Klausur WS 2006/2007 (10 Punkte = 10 min), Lösung

Aufgabe 5 Echtzeitbetriebssysteme **(10 Punkte)**

- TinyOS - ZebraNet: Auslegung auf Sensornetzwerke mit limitierter Stromversorgung.
- OsekTime - Zugsteuerung: Zeitgesteuertes Betriebssystem und damit klare Trennung einzelner Softwarekomponenten.
- QNX - Sicherheitsüberwachung: Hartes, stark skalierbares (kleiner Kernel) Echtzeitbetriebssystem.
- VxWorks - Fabrikanlagensteuerung: Hartes Echtzeitbetriebssystem mit guter Unterstützung unterschiedlichster Sensorik.
- WindowsCE - PDA: weiches Echtzeitbetriebssystem mit Schwerpunkt auf Benutzerfreundlichkeit (z.B. GUI), Kompatibilität mit Anwendungen von Standardbetriebssystemen