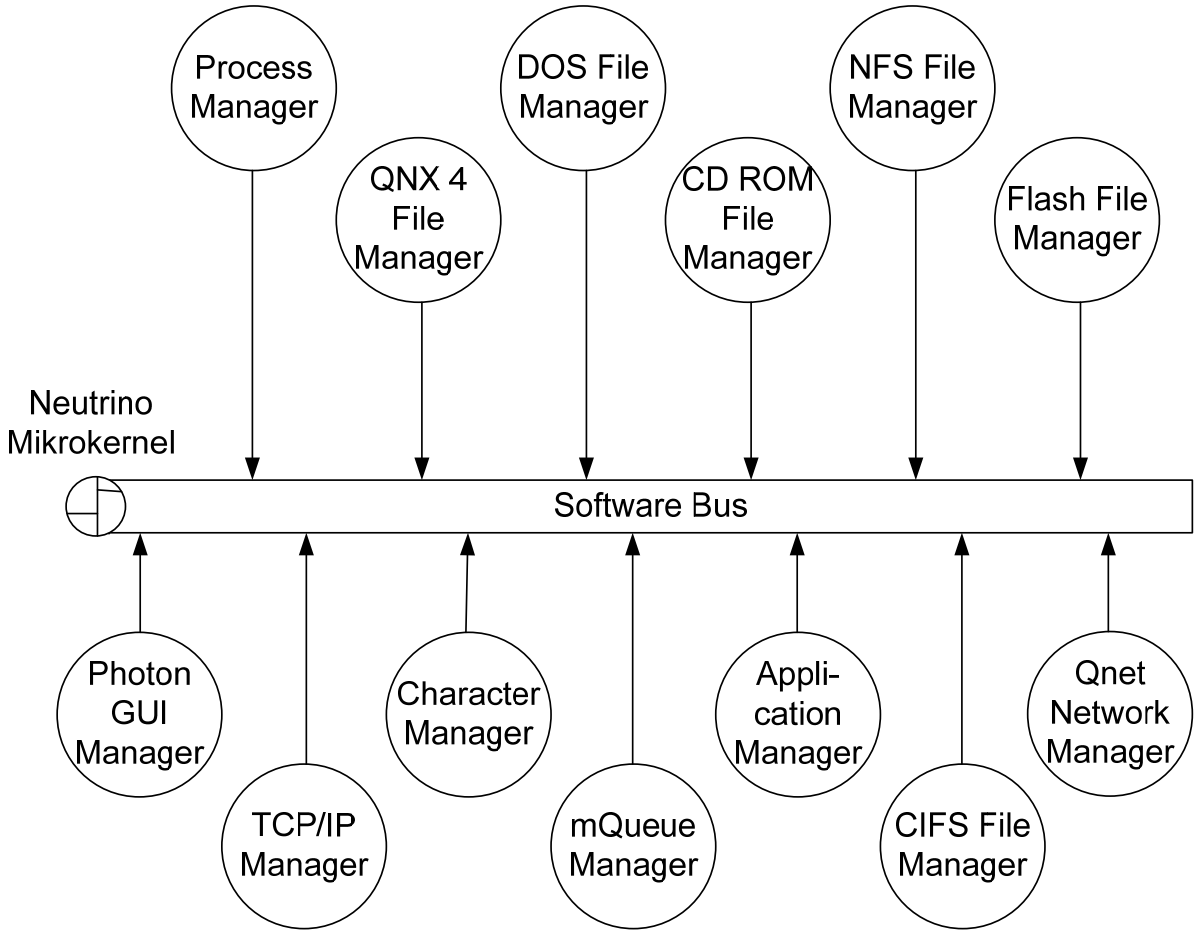
Echtzeitbetriebssysteme

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. $\Rightarrow$ 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

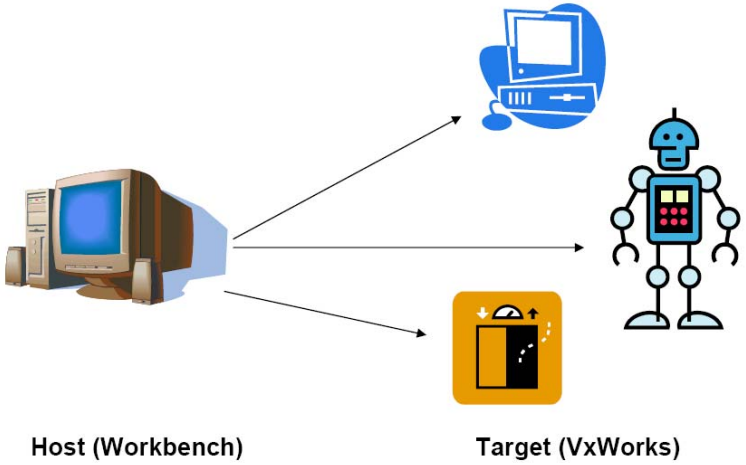
- Als wichtigster Prozess läuft in QNX der Prozessmanager.
- Die Aufgaben sind:
  - Prozessmanagement:
    - Erzeugen und Löschen von Prozessen
    - Verwaltung von Prozesseigenschaften
  - Speichermanagement:
    - Bereitstellung von Speicherschutzmechanismen,
    - von gemeinsamen Bibliotheken
   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

Echtzeitbetriebssysteme

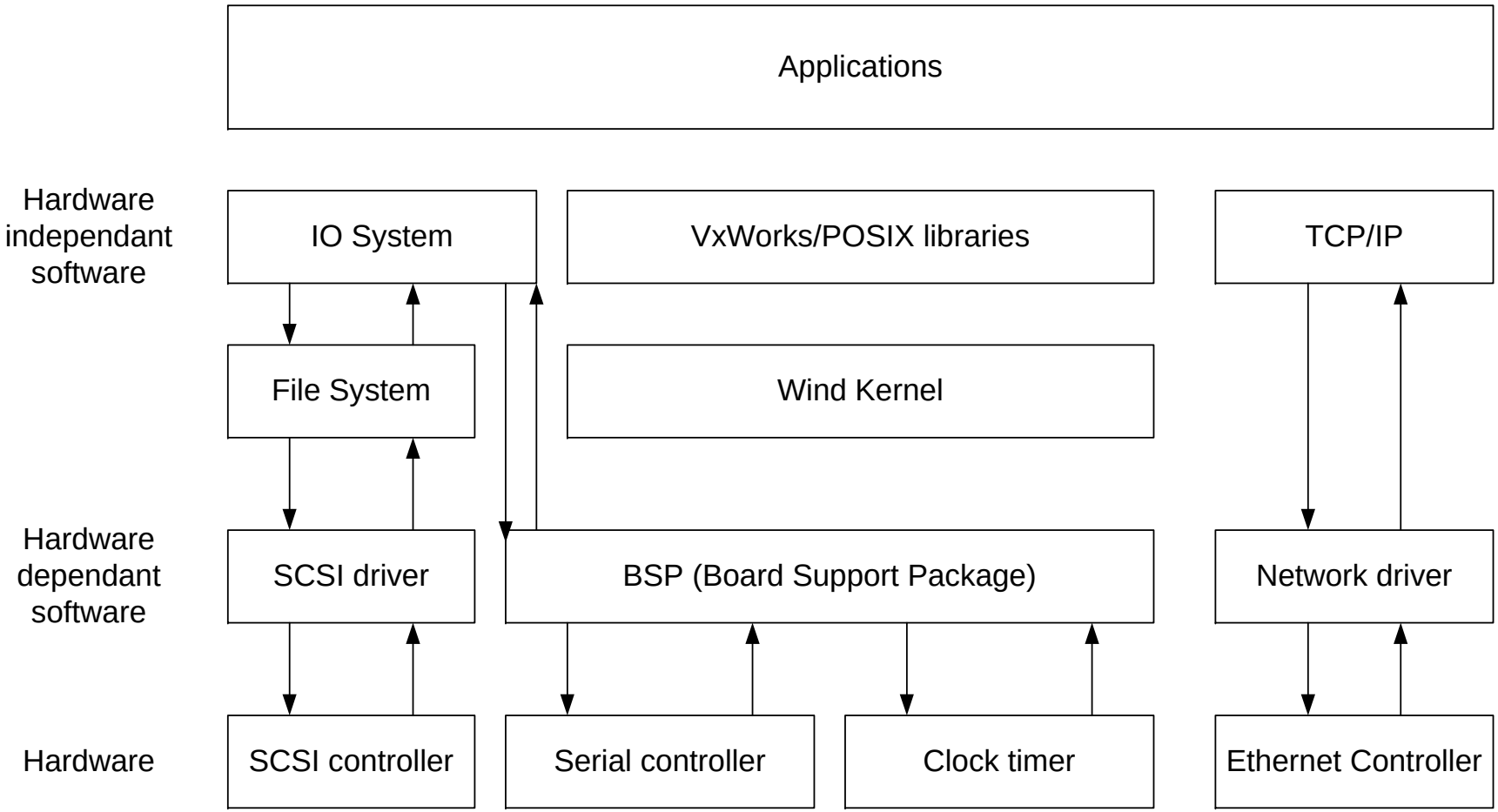
VxWorks

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, allerdings 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.
- **Uhrenauflösung**: Die Uhrenauflösung kann auf eine maximale Rate von ca. 30 KHz (abhängig von Hardware) gesetzt werden.
- **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

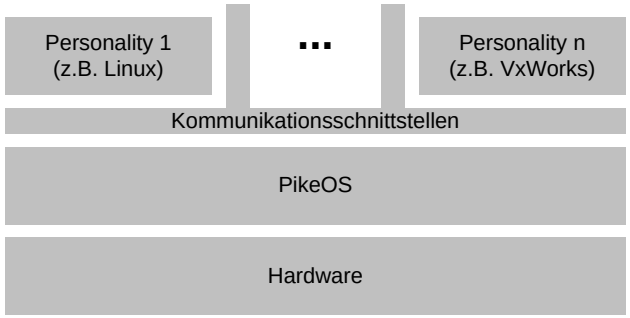
- Zur Interprozesskommunikation werden folgende Konzepte unterstützt:
  - Semaphor
  - Mutex (mit Prioritätsvererbung)
  - Nachrichtenwartschlangen
  - Signale
- Seit Version 6.0 wird zudem Speichermanagement angeboten:
  - Der Entwickler kann Benutzerprozesse mit eigenem Speicherraum entwickeln.
  -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



Architektur

- 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

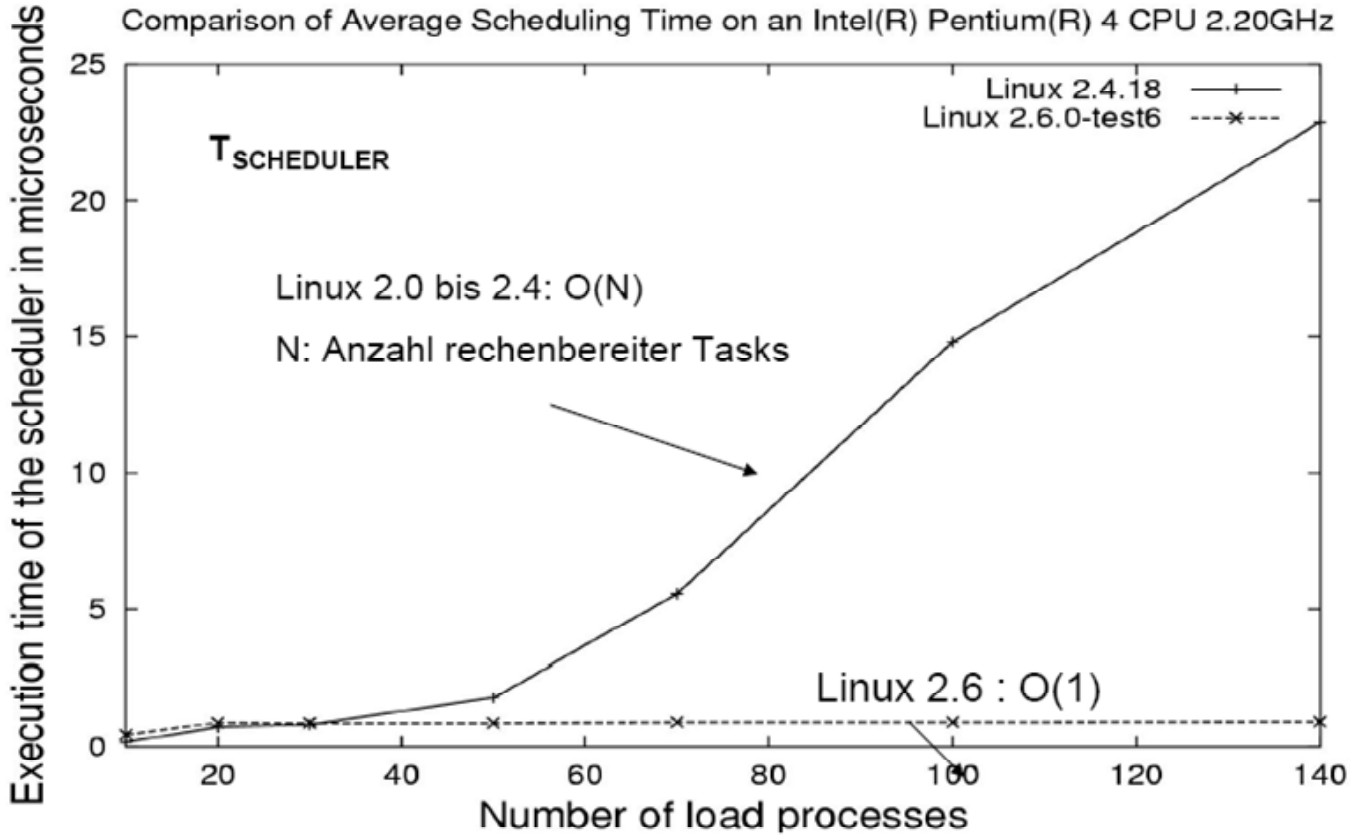
Echtzeitbetriebssysteme

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.
  $\Rightarrow$ 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

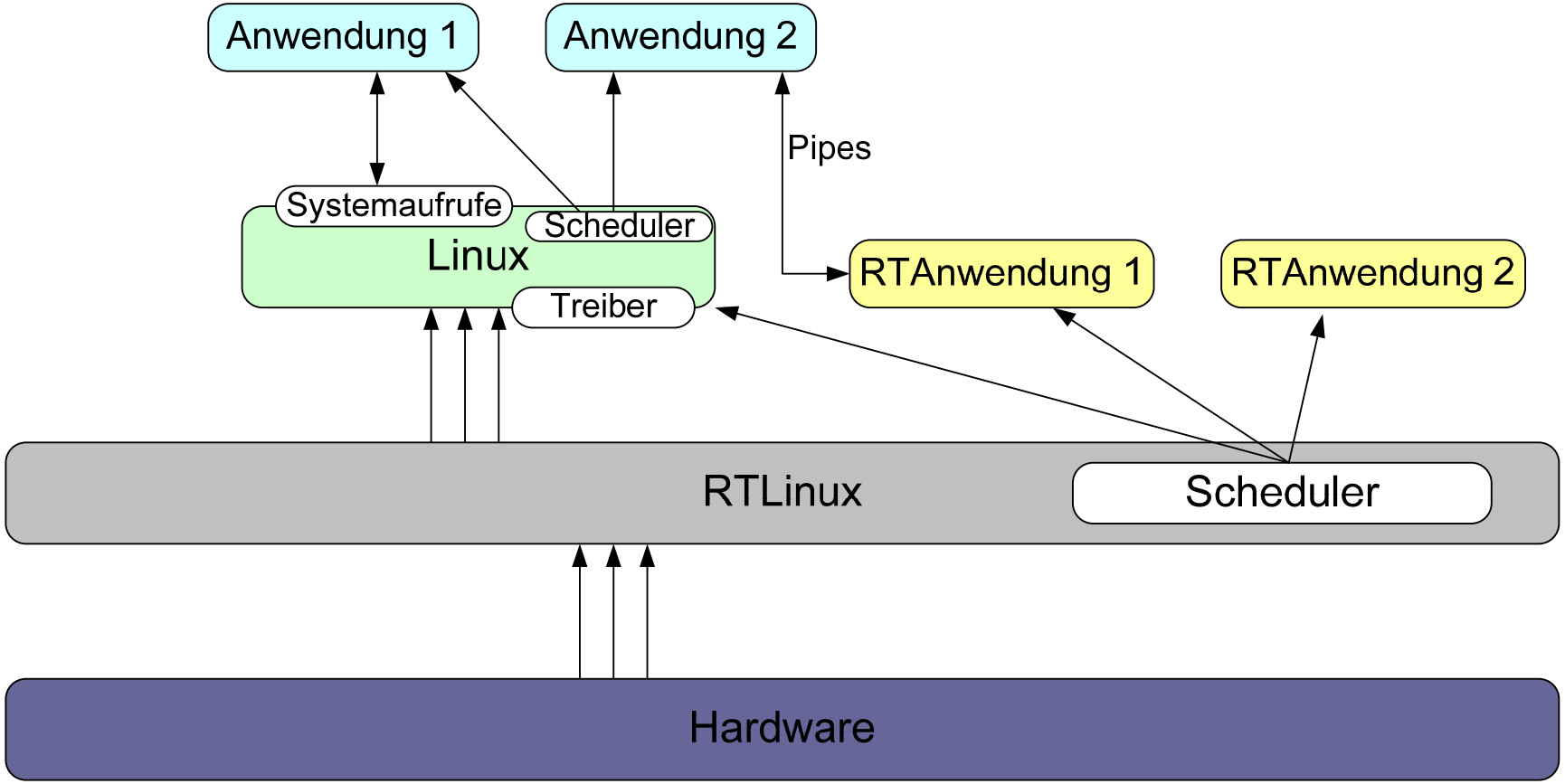
Motivation

- Aus diversen Gründen ist die Verwendung von Linux in Echtzeitsystemen erstrebenswert:
  - Linux ist weitverbreitet
  - Treiber sind sehr schnell verfügbar
  - Es existieren viele Entwicklungswerkzeuge $\Rightarrow$ 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 $\Rightarrow$ 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 $\Rightarrow$ 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

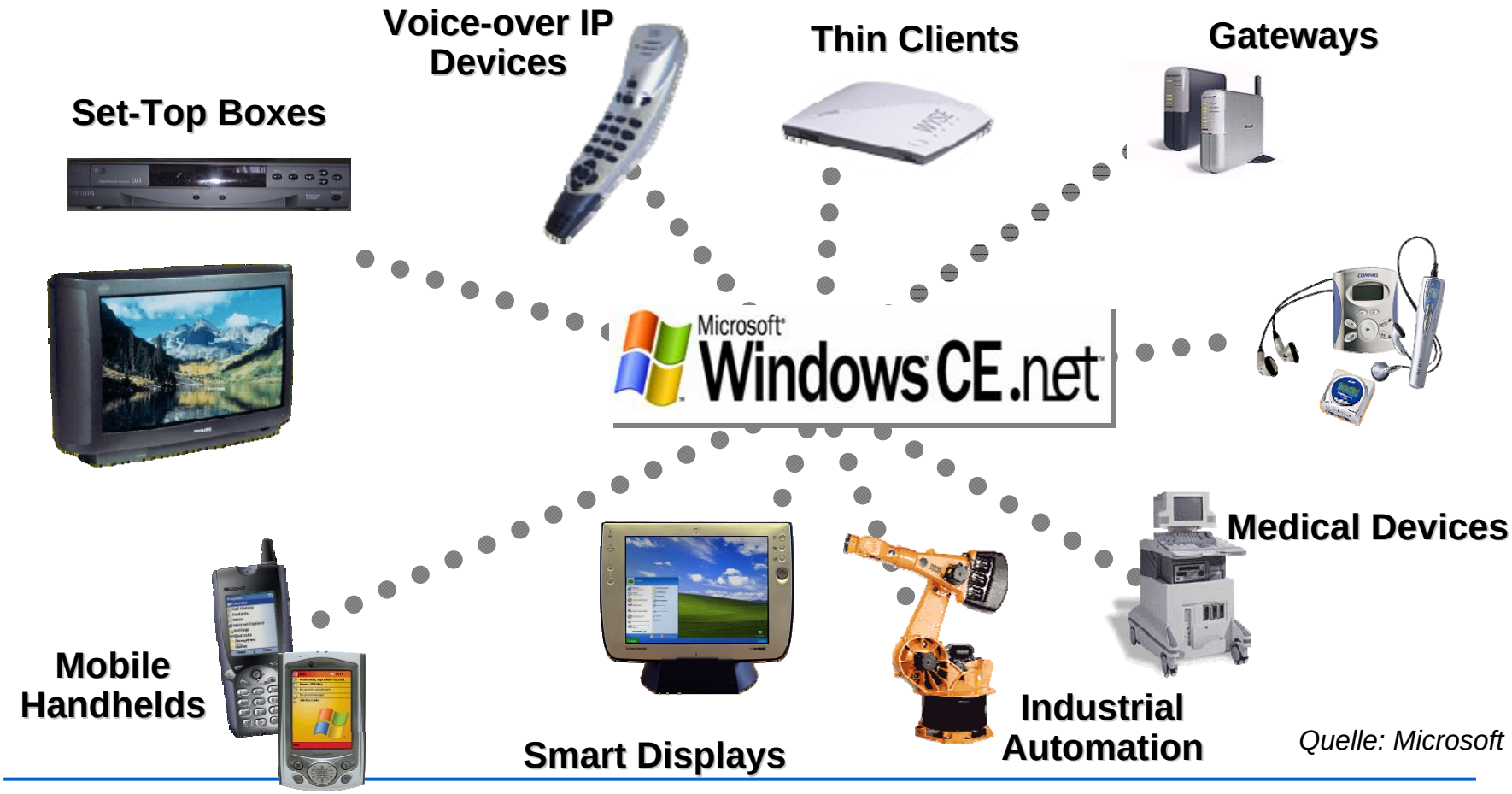
- Windows CE
  - 32-bit, Echtzeitbetriebssystem
  - Unterstützung von Multitasking
  - Stark modularer Aufbau
  - Skalierbar entsprechend der gewünschten Funktionalität
- Windows Embedded
  - „Skalierbares Windows XP“
  - Komponenten von XP können entfernt werden um den benötigten Speicherplatz zu minimieren

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

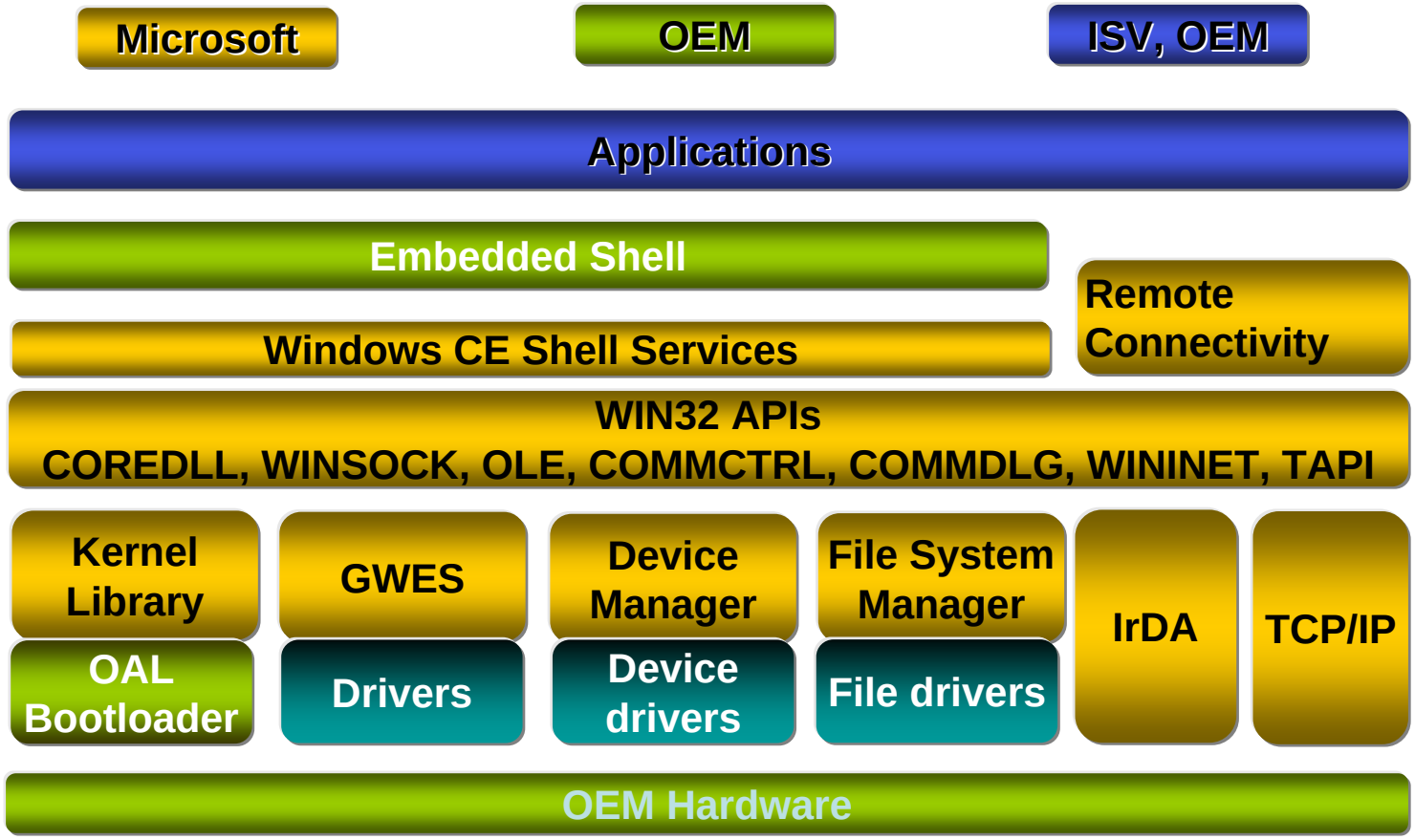
Quelle: Microsoft

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.