EMES: Eigenschaften mobiler und eingebetteter Systeme

Embedded- und RT-Betriebssysteme

Dr. Felix Salfner, Dr. Siegmar Sommer

Wintersemester 2010/2011

Aufgaben eines Betriebssystems I

- Prozeßmanagement
  - Erzeugung, Unterbrechung, Wiederaufnahme und Beendigung von System- und Nutzerprozessen, Prozeßsynchronisation, Prozeßkommunikation, Deadlock-Behandlung
- Dateimanagement (file management)
  - Erzeugung und Vernichtung von Dateien und Verzeichnissen, Möglichkeiten zur Manipulation von Dateien, Backup
- Speichermanagement
  - Wer nutzt welchen Teil des Speichers, welcher Prozeß kommt in den Speicher, Speicherreservierung, Management des virtuellen Speichers
- E/A-Management
  - Pufferungssystem, Bereitstellung einer allgemeinen E/A-Schnittstelle, Behandlung bestimmter Geräte

Aufgaben eines Betriebssystems II

- Schutzfunktionen (protection)
  - Management des Zugangs zum Rechner und dessen Ressourcen (accounting), Schutz der Prozesse vor den Aktivitäten anderer
- Kommunikation
  - Austausch von Informationen mit anderen Systemen (networking), Management entfernter Ressourcen
- Kommandoausführung
  - Schnittstelle zwischen Nutzer und System, Kommandoshell, Graphische E/A, ...

Betriebssysteme und Echtzeitsysteme

- Es gibt Echtzeitbetriebssysteme (z.B. vxWorks, LynxOS, QNX, rtLinux, PURE und weitere), aber . . .
- . . . braucht ein Echtzeitsystem zwangsläufig ein Betriebssystem?

Betriebssysteme und Echtzeitsysteme

- Es gibt Echtzeitbetriebssysteme (z.B. vxWorks, LynxOS, QNX, rtLinux, PURE und weitere), aber . . .

- . . . braucht ein Echtzeitsystem zwangsläufig ein Betriebssystem?
  - Nein, denn
    - Abarbeitung eines einzigen Programmes ist einfach
    - Abarbeitung einiger weniger (1,2. . . 5) ist einfach
    - Es existieren Echtzeitsysteme ohne Betriebssystem
  - Aber:
    - Steigende Anzahl von Programmen und erforderlicher grundlegender Funktionalität macht Aufwand immer größer
    - Betriebssystem bietet die grundlegenden Funktionen für alle Anwendungen an und vereinfacht diese damit

Echtzeitbetriebssysteme

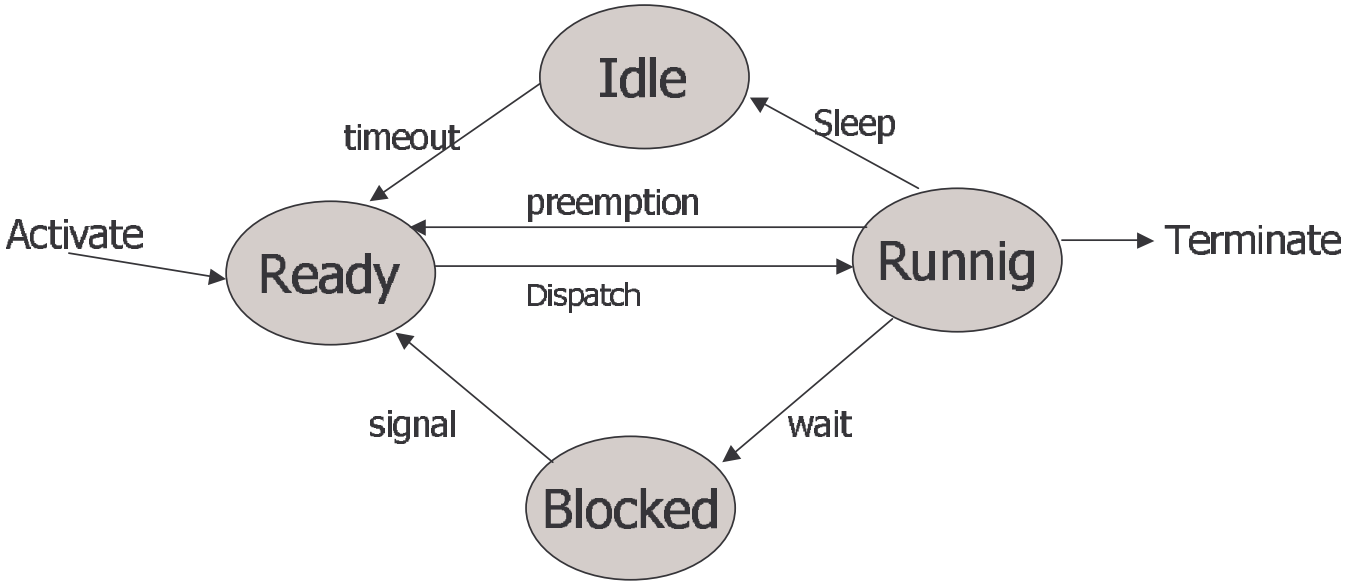
- Anforderungen sind andere als bei normalen Betriebssystemen
  - Statt Fairness wird vorhersagbares Verhalten benötigt - schnell und deterministisch auf Ereignisse reagieren
  - Statt hohem Durchsatz wird garantierter Durchsatz benötigt
  - Entwickler für Echtzeitverhalten der Applikation verantwortlich
  - Grundsätzliche Ansätze: Minimale Interrupt-Latenz, minimale Zeit für Kontextwechsel
  - Adaptierbarkeit auf spezielle Probleme ist bedeutsam
- Wiederverwendung von Ideen und Konzepten nur begrenzt möglich
- Hier: Betrachtung der grundlegenden Funktionen in Bezug auf die Anwendung bei Echtzeitsystemen

Grundlegende Funktionen eines Echtzeit-Betriebssystems

- Prozeßmanagement bzw. Taskmanagement
- Speichermanagement
- Interrupt-Behandlung
- Behandlung von Programmausnahmen
- Prozeßsynchronisation
- Zeit-Management
- CPU Scheduling

Taskmanagement

- Tasks haben zusätzliche Eigenschaften
  - (Vorgegebene) Prioritäten
  - Deadlines (hart, weich, IRIS)
  - Periodizität: periodisch, aperiodisch, sporadisch
  - Perioden bzw. minimal interarrival Zeiten
  - nichtunterbrechbar/unterbrechbar
- Taskverwaltung wie "üblich":



Interrupt-Behandlung

Wie in Nicht-Echtzeitbetriebssystemen, aber:

- Bekannte und begrenzte Verzögerungen für Interrupt-Behandlung
- Ziel: Kurze Interrupt-Behandlungsroutinen
- Interrupts sind mit Tasks assoziiert — Interrupt aktiviert Task

Speichermanagement

- Standard-Methoden:
  - Block-basiert
  - Paging/ Swapping
- Kein virtueller Speicher für harte Echtzeit-Tasks
  - alle Speicherseiten der Task im Speicher “locken”
  - Memory-Pinning

Behandlung von Programmausnahmen I

- Ausnahmesituationen (Speicher voll, Deadlock, Timeouts) müssen behandelt werden
- Unterschiedliche Ebenen
  - Fehler auf Systemebene, z.B. Deadlock
  - Fehler auf Taskebene, z.B. Timeout
- Standard-Techniken:
  - Systemrufe mit Fehlercode (muß vom Programmierer aber auch benutzt werden)
  - Watchdog
- Alle Szenarios müssen abgedeckt werden (kompliziert!)
  - Auslassen eines möglichen Falles kann zur Katastrophe führen!

Behandlung von Programmausnahmen II

- Überwachung des Ablaufes durch unabhängige Instanz
- Hardware-Watchdog
  - Spezielle Hardware überwacht System und reagiert auf erkannte Fehler, z.B. Ausbleiben von Alive-Signalen
  - Nur für wenige Einsatzfälle benutzbar, teilweise hoher Aufwand
  - Kann auch bei Komplettabstürzen der Software reagieren
- Software-Watchdog
  - Task mit hoher Priorität parallel zu anderen Tasks
  - Überwacht Systembedingungen (z.B. Timeouts, Invarianten, Wertebereiche)
  - Löst Aktionen bei Verletzung aus
  - Kann nicht reagieren, wenn das OS abgestürzt ist

Task Synchronisation

- Übliche Techniken
  - Semaphore
  - Gemeinsame Variablen
  - Gemeinsame Puffer
  - Mailboxen
  - Message Passing
  - Signale
- "Bekannte" Probleme:
  - Gefahr von Deadlocks — ausgiebig in entsprechenden Algorithmen behandelt
- Aber: Neue Probleme, die es in Nicht-RTOS nicht gibt
  - Prioritäteninvertierung durch Blockade infolge der Nutzung von Ressourcen
  - Lösung: Priority Inheritance und Priority Ceiling

Zeit-Management

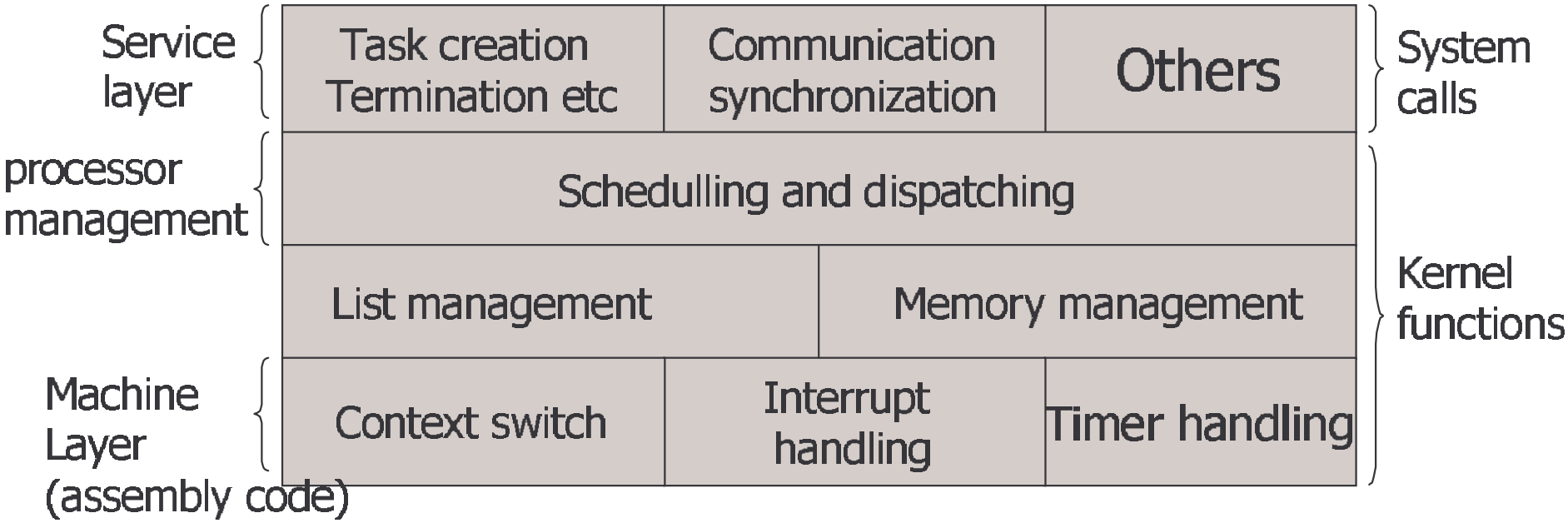
Timer-Ticks repräsentieren den Ablauf der Echtzeit im System

- Üblicherweise Ticks von 1..50 ms
- Üblicherweise Erzeugung mit einer Hardware, die pro Tick einen Timer-Interrupt auslöst
- Interrupt-Behandlung zählt Systemzeit hoch
- Ticks können (manchmal) in der Länge an die Parameter der Anwendungen angepaßt werden
- Alle Zeiten im System sind Vielfaches des Timerticks
- Auflösung und Bitbreite des Timers müssen beachtet werden — z.B. 32 Bit Timer mit 1 ms Auflösung läuft nach 50 Tagen über

CPU-Scheduling

- Überführung von Tasks aus READY nach RUNNING
- Aufbau eines Schedules unter Benutzung von Schedulingverfahren
  - EDF
  - RMA
  - . . .
- Details: Vorlesung über Scheduling

Beispielarchitektur eines RT-Kernels



Garantien

Ein Echtzeitbetriebssystem soll garantieren:

- Bekannte Ausführungszeiten für Systemrufe
- Begrenzte Blockadezeiten für Zugriff auf gemeinsame Ressourcen
- Begrenzte Delays für Interrupt-Behandlung
- Bekannte Ausführungszeit für Kontext-Switch
- Einhaltung aller Deadlines (für schedulbaren Taskset)
- Globale Zeit für alle Tasks

Time Driven RT-OS

- Ein fester Zeitplan wird vor der Laufzeit konstruiert, der zur Laufzeit nur noch abgearbeitet wird
- Hohes A-priori-Wissen
- Sehr geringer Laufzeitaufwand
- Keine Synchronisation nötig, da vorher berechnet
- Scheduling beschränkt sich auf Abarbeitung des Zeitplanes
- Reaktionen sind nur auf eingeplante Ereignisse möglich
- Umgebungsdaten werden im Polling gelesen (Abfrage alle n Zeiteinheiten laut Zeitplan)
- Hohe Auslastung erzielbar
- Flexibilität gegenüber Änderungen gering

Event Driven RTOS

- Tasks “erscheinen” periodisch, aperiodisch oder sporadisch (lösen dabei meist einen Interrupt aus)
- Scheduler fügt sie in READY-Queue ein
- Scheduler entscheidet dynamisch über Reihenfolge der Abarbeitung und implementiert Schedulingverfahren

Praxis:

- Oft Mischung beider Techniken

Features moderner RTOS

- Multitasking
- Prioritätenbasiertes-Scheduling
  Anwendungen müssen entsprechend entwickelt werden!
- Schnelle Reaktionen auf externe Interrupts
- Mechanismen für Prozeßkommunikation und -synchronisation
- Kleiner Kernel (bis zu Mikrokernel)
- Schneller Kontextswitch
- Echtzeituhr als interne Zeitreferenz

Mikrokernel

- Minimaler Code im privilegierten Kernmodus
  - Speicherverwaltung
  - CPU-Verwaltung
  - Interprozesskommunikation
  - Trennung von Mechanismen (Kern) und Policies (User-Mode)
- Alles andere als 'Server': Treiber, Protokolle, Dateisysteme, ...
- Scheduler und Timer aus Effizienz häufig auch im Kern

Kategorien existierender RTOS

- Prioritätenbasierte Kernel für eingebettete Applikationen
  - OSE, VxWorks, VRTX32,pSOS, LynxOS
- RT-Erweiterungen zu existierenden Nicht-RTOS durch Memory-Locking, Scheduling-Server, o.ä.
  - rtLinux, Real-Time Windows NT, Real-Time Mach
- Forschungs-RT-Kerne
  - MARS, Spring
- Laufzeitsysteme für RT-Programmiersprachen
  - Ada, Erlang

Gemeinsamkeiten I

- Konformität zu Standards (POSIX-RealTime-API)
- Modularität und Skalierbarkeit
  - Kleiner Kern
  - Konfigurierbares OS
  - Von ROM-Systemen bis zu großen verteilten Systemen
- Geschwindigkeit und Effizienz
  - Niedriger Overhead
  - Nachrichten senden ohne Kontext-Switch
  - Geringe Zeiten für Kontext-Switch, Interrupt Latency
- Hochoptimierter Code für nichtpreemptive Teile von Systemrufen
- Geteilte Interrupt-Behandlung (kleiner Teil für nichtpreemptive Routinen und unmittelbare Behandlung, Ausführung der verbleibenen Arbeit mit entsprechenden Prioritätslevel)

POSIX RT API

- OpenGroup - Portable Open System Interface + X
  - POSIX 1 - Grundlegende Systemaufrufe (fork, read, write, ...)
  - POSIX 1.c - Thread-Erweiterungen (pthread_create, ...)
- POSIX 1.b - Echtzeiterweiterungen
  - Offene Spezifikation - http://tinyurl.com/yaayjro
  - Prioritätsbasiertes Scheduling, Echtzeitsignale, Semaphoren, Nachrichtenkommunikation, gemeinsamer Speicher, Sperren von Speicher, Asynchrones I/O, Timer
- In Teilen (rtLinux) oder vollständig (LynxOS) vom Echtzeitbetriebssystem implementiert

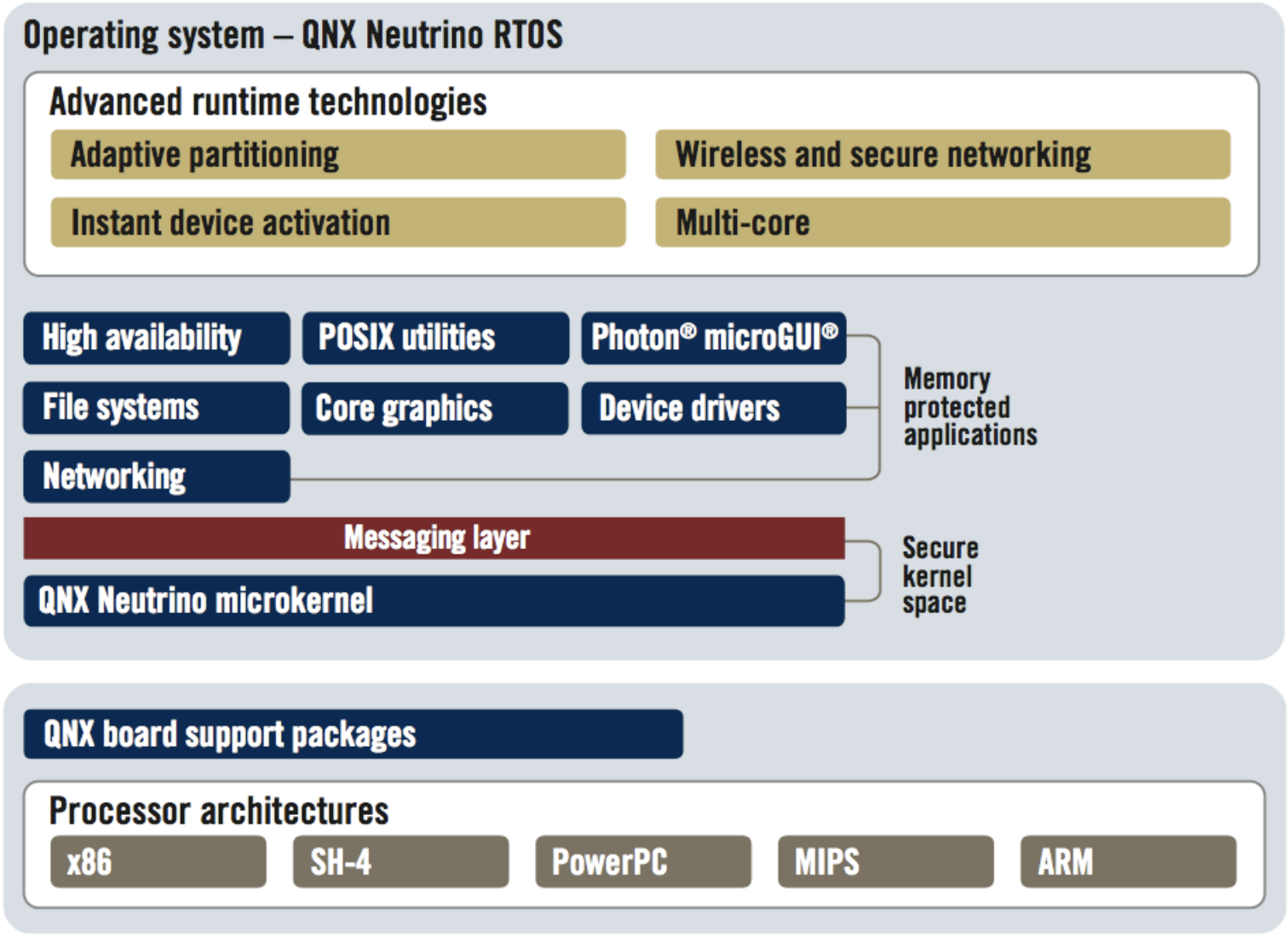
Gemeinsamkeiten II

- Scheduling
  - Wenigstens 32 Prioritätenebenen
  - Unterstützung für Round-Robin, FIFO und nutzergesteuerte Änderung von Prioritäten
- Priority Inheritance, manchmal Priority Ceiling (beides abschaltbar)
- Timer: Auflösung bis zu Nanosekunden (nicht immer sinnvoll, weil Timer-Interrupt bis zu einer Mikrosekunde dauert)
- Memory Management
  - Protection (u.U. konfigurierbar auf mehreren Ebenen)
  - Kein Paging (bzw. abschaltbar)
- Netzwerk: Unterstützung für TCP/IP, Streams,...

QNX/Neutrino

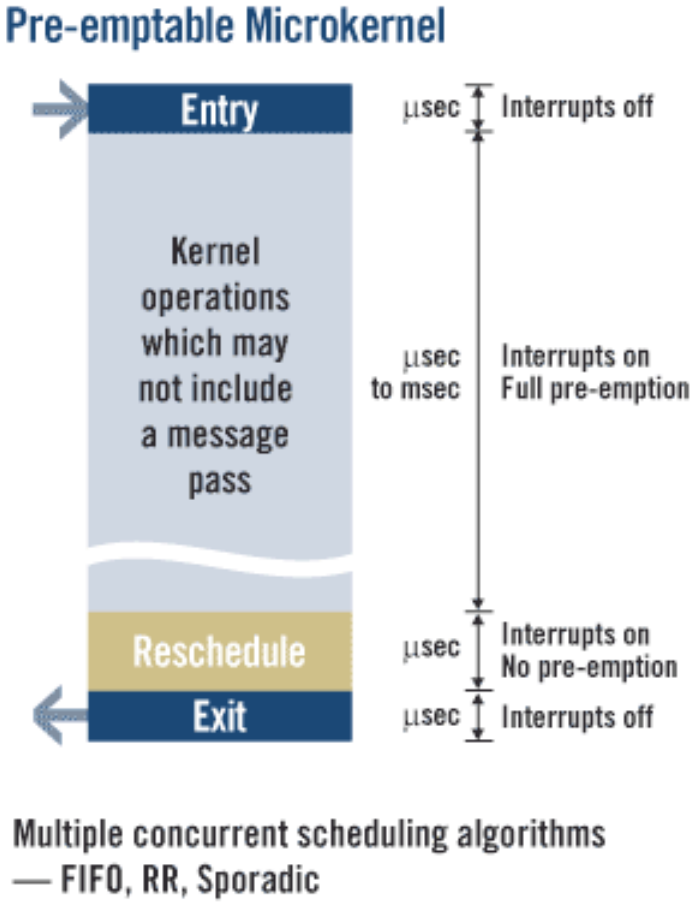
- Echter Mikrokernel, bietet Thread- und Echtzeit-Services
- Resource Manager bieten weitere Services an (u.a. auch Prozessbehandlung, da Kern nur Threads kennt ohne Speicherschutz)
- Optionale Teile können zur Laufzeit ein- und ausgeschlossen werden
- System kann bis zu 12K klein sein (Größe des Mikrokernels)
- Message-passing OS (Nachrichten mit Prioritäten sind Mittel der Kommunikation zwischen allen Threads)
- Andere Kommunikation (z.B. POSIX-Message-Queues) werden außerhalb des Kerns auf QNX Message-Passing abgebildet
- Vorhandensein atomarer Funktionen für Addition, Subtraktion, Bitsetzen, Bitlöschen und Bitkomplement

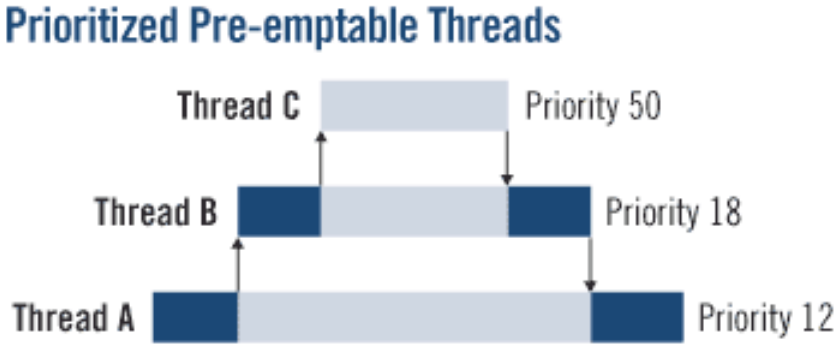
QNX/Neutrino

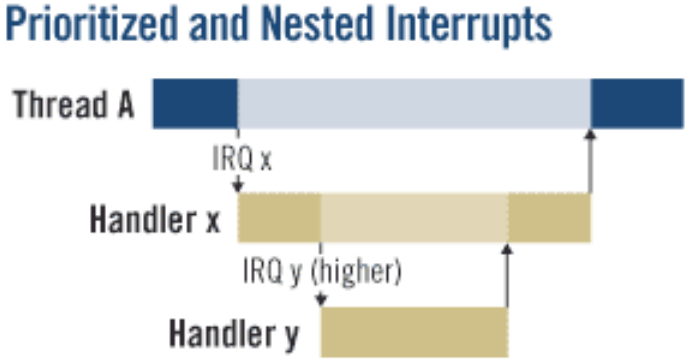


QNX Interrupt Handling

- Garantierte Latenzzeiten für hochpriore Interrupts







LynxOS

- Firma LynxWorks, gegründet 1986
- 28 K Mikrokernel, support for Motorola / 386 / ARM / PowerPC
- POSIX / Linux ABI kompatibel
- Kernel-Plugins für Services (I/O, Filesystem, TCP/IP)
- Kernel-Plugins sind multithreaded
- Kein Kontext-Switch beim Senden einer Nachricht an ein Kernel-Plugin
- Kommunikation zwischen Plugins braucht nur einige Instruktionen
- Paging bei Bedarf

pSOSystem

- Versionen für Ein- und verteilte Mehrprozessorsysteme
- Objekt-orientiertes System
- Objekt-Klassen beinhalten Tasks, Speicherbereiche, Nachrichten-Queues, Semaphore
- Objekte können lokal (nur auf dem eigenen Prozessor) und global (von jedem Prozessor aus benutzbar) sein
- Gerätetreiber sind nicht Teil des Kerns, sondern ladbare Module
- Interrupts führen zum direkten Ansprung der Interruptroutine ohne Kernelbeteiligung

VRTX

- Zwei Kerne
  - VRTXsa für Performance
  - VRTXmc für Speicher- und Energieeffizienz
- Genügt Standards der FAA (Federal Aviation Agency) für sicherheitskritische Software an Bord von Flugzeugen
- Eingesetzt in der MD-11
- Bietet “Hooks” für Erweiterungen
- Applikationen können Systemrufe hinzufügen

VxWorks

- Erweiterung zu VRTX
- Monolithisches System
- Cross-Development (Workbench IDE, früher Tornado)
- Virtuelle Speicherverwaltung, Standard ist ein gemeinsamer Speicherraum
  - Für mehrere CPUs
  - Cache-Konfigurierbarkeit (z.B. kein Cache für Speicher mit DMA-Zugriffen)
- (abschaltbare) Priority Inheritance

- Betriebssystem von “Pathfinder”