9 Echtzeit-Betriebssysteme

Wichtige Eigenschaften gegenüber Universalrechnern herausgegriffen

9.1 Anforderungen

- stabiler Betrieb rund um die Uhr
- definierte Reaktionszeiten
- parallele Prozesse
- Unterstützung von Mehrprozessorsystemen
- schneller Prozeßwechsel (geringer Prozeßkontext)
- Unterbrechungshierarchie
- Unterbrechungsbehandlungen in Benutzerprozessen (Unterbrechungs–Antwortprogramme)
- Prioritäten für Prozesse
- Scheduling streng präemptiv nach Prioritäten mit FIFO bei gleicher Priorität
- Warteschlangen im System nach Prioritäten
- Prioritätsvererbung

- umfangreiche Zeitdienste
  - ⋆ absolute und relative Uhren
  - ⋆ Weckdienste (delay)
  - ⋆ Prozesse einplanen mit Zykluszeit
- Zeitüberwachung aller Wartezustände (timeout)
- Prozeß–Management (Task–M.)
  - ⋆ erzeugen, starten, löschen
  - ⋆ anhalten, wiederaufnehmen; abhängig von Zeit-punkten, -intervallen, anderen Ereignissen, periodisch
- Synchronisation und Kommunikation
  Semaphor, Botschaften (n:m), Ereignisse, Monitore (protected objects), Rendezvous (jeweils einfache Konzepte, mit Prioritäten)
- einfaches Speicher–Management
- keine virtuelle Adressierung
- wichtige Prozesse speicherresident

- Direkter Ein–/Ausgabeverkehr mit Peripherie
  - ⋆ vielfältigste Peripherie
  - ⋆ direkter Zugriff auf Hardware-Adressen und -Register durch den Benutzer
  - ⋆ Treiber in Benutzerprozessen
    möglichst schnell und einfach zu implementieren
    dynamisches Binden an Systemkern
  - ⋆ direkte Nutzung DMA
  - ⋆ keine mehrfachen Puffer: direkt vom Benutzerpuffer auf das Gerät
- Einfachste Dateistrukturen
- Protokoll für Feldbus oder LAN–Bus, möglichst hardware-unterstützt
- statt Programmierschnittstelle ein Bediensystem (Kommandos, Zustandsübersicht, Messungen)
- Generierung maßgeschneiderter BS aus Bibliotheken (besonders vorteilhaft für Masseneinsatz in eingebetteten Systemen)

9.2 Beispiele für Produkte

- unvollständige Liste!
- viele herstellerspezifische (ad-hoc) Lösungen für bestimmte Produktbereiche
- Einsatzgebiete
  - ⋆ Industrie-Steuerungen
  - ⋆ Prozeßvisualisierung
  - ⋆ Informationssysteme
  - ⋆ Mobile Computer
  - ⋆ Consumer electronics
  - ⋆ Multimedia-Anwendungen
  - ⋆ Hausgeräte
  - ⋆ Unterhaltungsindustrie
  - ⋆ Kommunikationssysteme
  - ⋆ Handys
  - ⋆ eingebettete Systeme, u.a. Automobil
- Tendenz zu universelleren (firmen- und produktübergreifenden) Echtzeitsystemen
  - ⋆ geringere Entwicklungs- und Wartungskosten
  - ⋆ Kompatibilität zu anderen

  - ⋆ geringerer Schulungsaufwand
  - ⋆ weit höhere Funktionalität und Qualität
- Schnittstelle typischerweise als 1:1-Systemaufrufe in der Programmiersprache (z.B. Fortran, Pascal, C, C++, Java)
- Schnittstelle variiert bei den Realzeitsystemen beträchtlich (wenig portabel)
- VRTX (32)
  VERSATILE REALTIME EXECUTIVE ("Vertex") von READY SYSTEMS für Motorola, Intel u. a.
- VRTXmc von Microtec Research
- VME_{exec}
  von Motorola für VME-Bus-Umgebung
- OS-9 von Dr. Keil für Motorola, Intel
- FLEXOS
  von DIGITAL RESEARCH für PC
- OSEK/VDX (Automobilhersteller)
- RMOS3 bei Siemens Automation & Drives
- QNX-RTOS von QNX Software Systems für PC

- VxWorks von Wind River für alle
- LynxOS
  von Lynx Real-Time Systems für Motorola, Intel, Sparc
  (Einsatz im Prozeßrechner–Praktikum)
- Windows-CE und Windows NT Embedded von Microsoft für eingebettete Systeme
  - ⋆ in D u.a. Partnerschaft mit Siemens dafür
- RT-Linux in vielen Varianten

9.3 Schichtenmodell

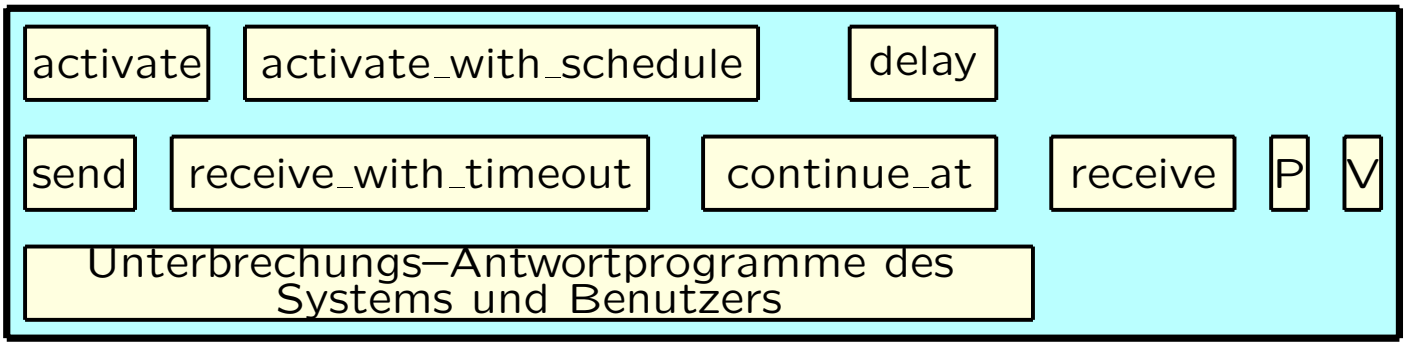
- erste Spezifikation in European Workshop on Industrial Computer Systems (EWICS) TC8 on Real-Time Operating Systems (1976-1979)
- für ein verteiltes Mehrprozessor-Echtzeit-Betriebssystem

  level 1: Context Switching,
  Basic Synchronisation,
  List Handling

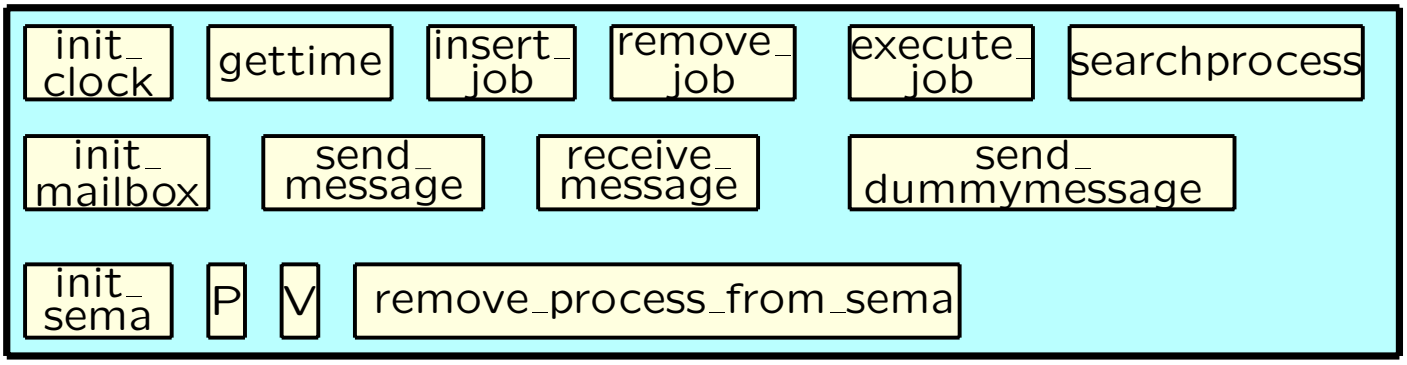
  level 2: Dispatcher Primitives

  level 3: Synchronization,
  Task Management
- Implementierung als Diplomarbeit 1981 auf Mehrrechnersystem (drei LSI11) am Lehrstuhl
- in den folgenden Skizzen:
  - ⋆ Komponenten unvollständig
  - ⋆ Komponenten durch einen typischen Dienst charakterisiert
  - ⋆ Beispiel, kein reales System als Modell

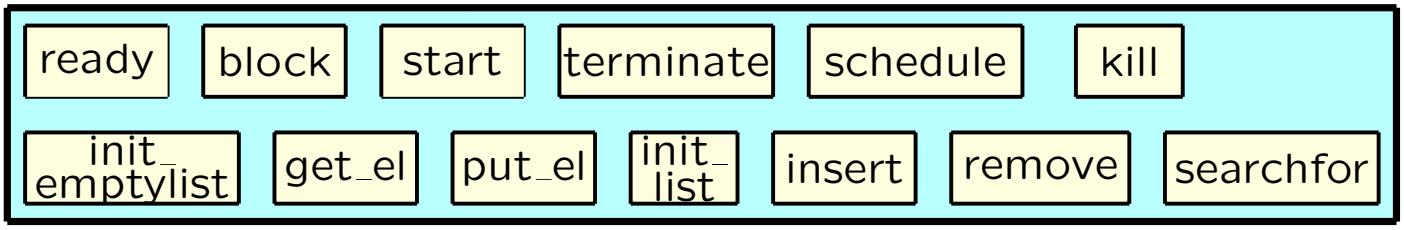
Schicht 4: Prozeduren für Systemdienste



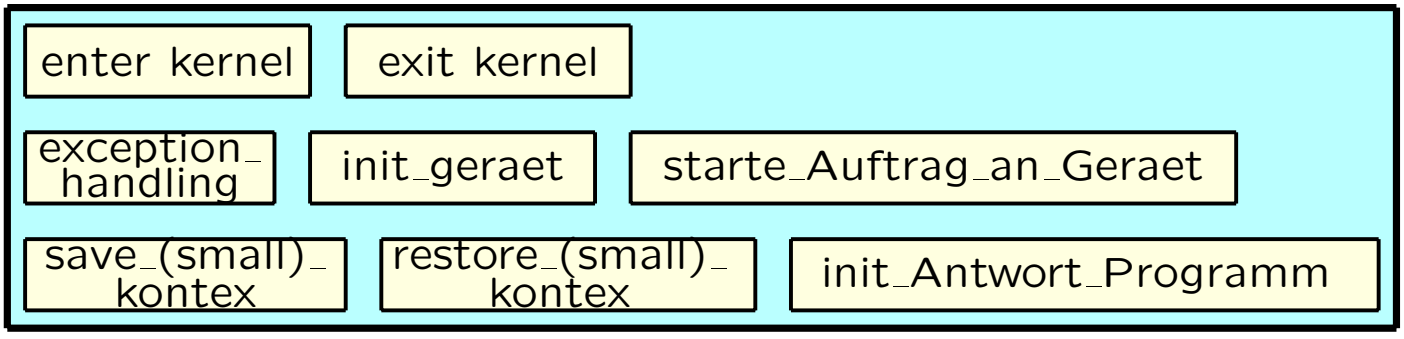
Schicht 3: Zeit, Botschaft, Semaphor



Schicht 2: Prozeßzustandsübergänge, Listenprozeduren



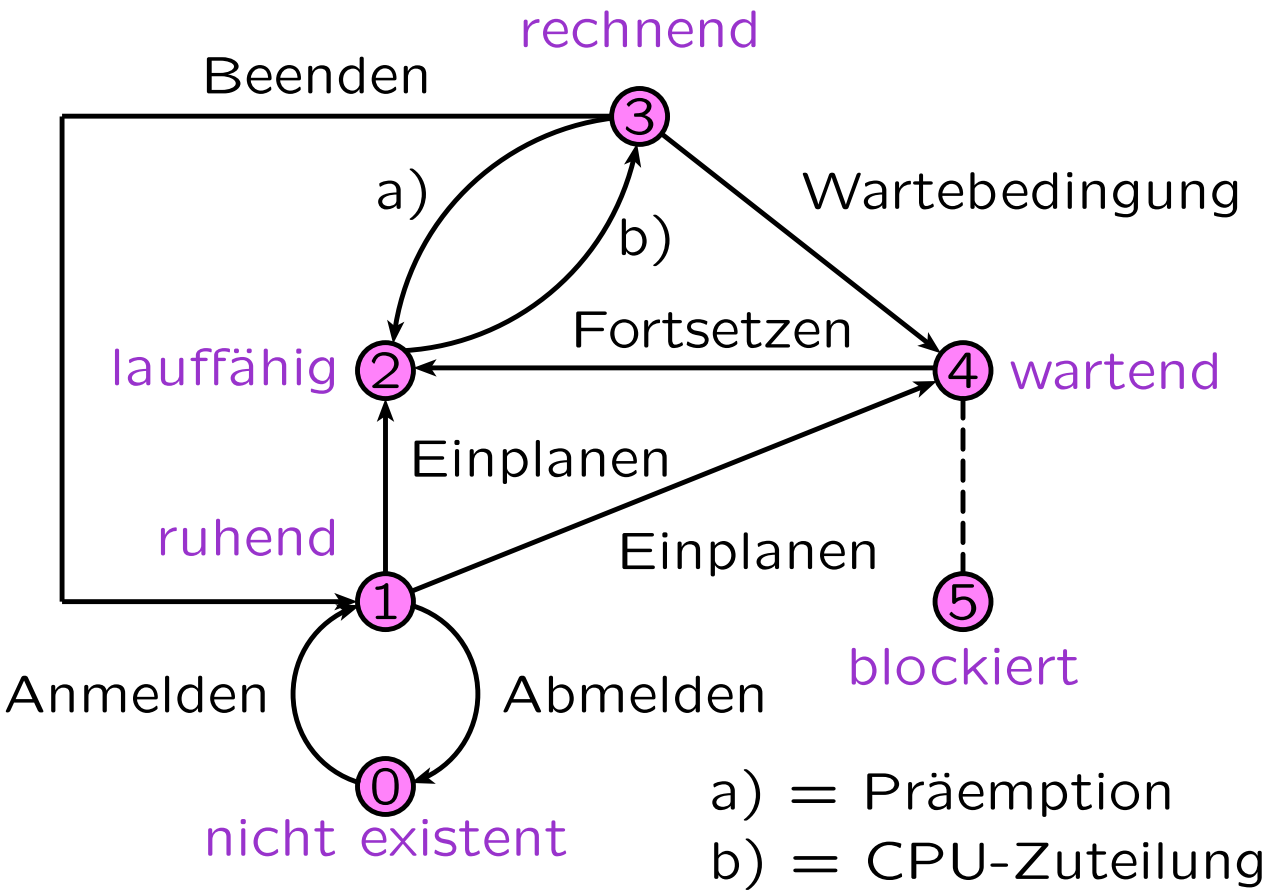
Schicht 1: Hardware–Abdeckung



9.4 Rechenprozesse (tasks, threads)

- Gleichzeitige physikalische Vorgänge erfordern parallele Aktionen im Mehrprogramm-, Mehrprozessor- oder Mehrrechnerbetrieb
- Prozeßrechner–Betriebssysteme müssen mehrere Prozesse präemptiv (unterbrechbar) nach Prioritäten steuern
- Unterbrechungs–Antwortprogramme und Betriebssystem–Kerndienste sind spezielle Abläufe, die ebenfalls wie Prozesse unterbrechbar
- Leichtgewichtsprozesse (threads)
  - ⋆ ohne eigenen Prozeßadreßraum
  - ⋆ laufen im Prozeßadreßraum eines (Schwergewichts-)Prozesses
  - ⋆ Prozeßwechsel billig
  - ⋆ Erzeugen threads billig
  - ⋆ schneller Zugriff auf gemeinsame Daten im gemeinsamen Adreßraum

- Zustandsdiagramm



- ruhender (inaktiver) Rechenprozeß (Zustand 1): keine Einplanung im BS, aber angemeldet und in den Listen des BS geführt; kein Auftrag zur Abwicklung
- Rechenprozeß aktiv in Zuständen > 1; Prozeß gestartet bzw. Start im BS eingeplant
- Zyklische Prozesse sind dauernd aktiv

- Zustand lauffähig (ready):
  Alle Voraussetzungen für Start bzw. Fortsetzung erfüllt;
  wartend auf Prozessorzuteilung
- Zustand wartend (waiting):
  Alle Betriebsmittel außer CPU zugeteilt, aber Prozeß wartet auf Eintreten einer Bedingung
  (z.B. Ablauf einer Wartezeit,
  Eintreffen eines Interrupts,
  Eingang einer Botschaft,
  Freigabeoperation (V-Operation) eines anderen Prozesses
- Zustand blockiert (blocked):
  Dem Rechenprozeß fehlt ein vom BS verwaltetes Betriebsmittel,
  z.B. der Arbeitsspeicher (nur für Hintergrundaufgaben, zeitkritische Prozesse dauernd aktiv und speicherresident)

- Prozeßaktivierung
  - ⋆ Variante 1:
    - ◦ Hauptprozess (maintask) wird gestartet
    - ◦ Prozesse ruhend und bei Bedarf durch Betriebssystemaufrufe aus anderen Prozessen aktiviert
    - ◦ für Meß- und Steuerungsaufgaben mit wechselnden Aktionen
  - ⋆ Variante 2:
    - ◦ alle Prozesse einmalig bei der Initialisierung aktiviert
    - ◦ keine dynamisch erzeugten Prozesse
    - ◦ Prozesse dauernd aktiv mit internem Zyklus von Arbeitsphasen und von Wartephasen auf Nachrichten
    - ◦ für Dienstprozesse (server), die auf Aufträge anderer Prozesse warten
  - ⋆ Variante 3 (in ADA):
    - ◦ Task wird innerhalb Blockstruktur deklariert und beim Durchlaufen der Deklaration automatisch gestartet
    - ◦ aber auch Taskvariable zur dynamischen Erzeugung

⋆ Beispiel zu Variante 1: (PEARL; Process and Experiment Automation Realtime Language)

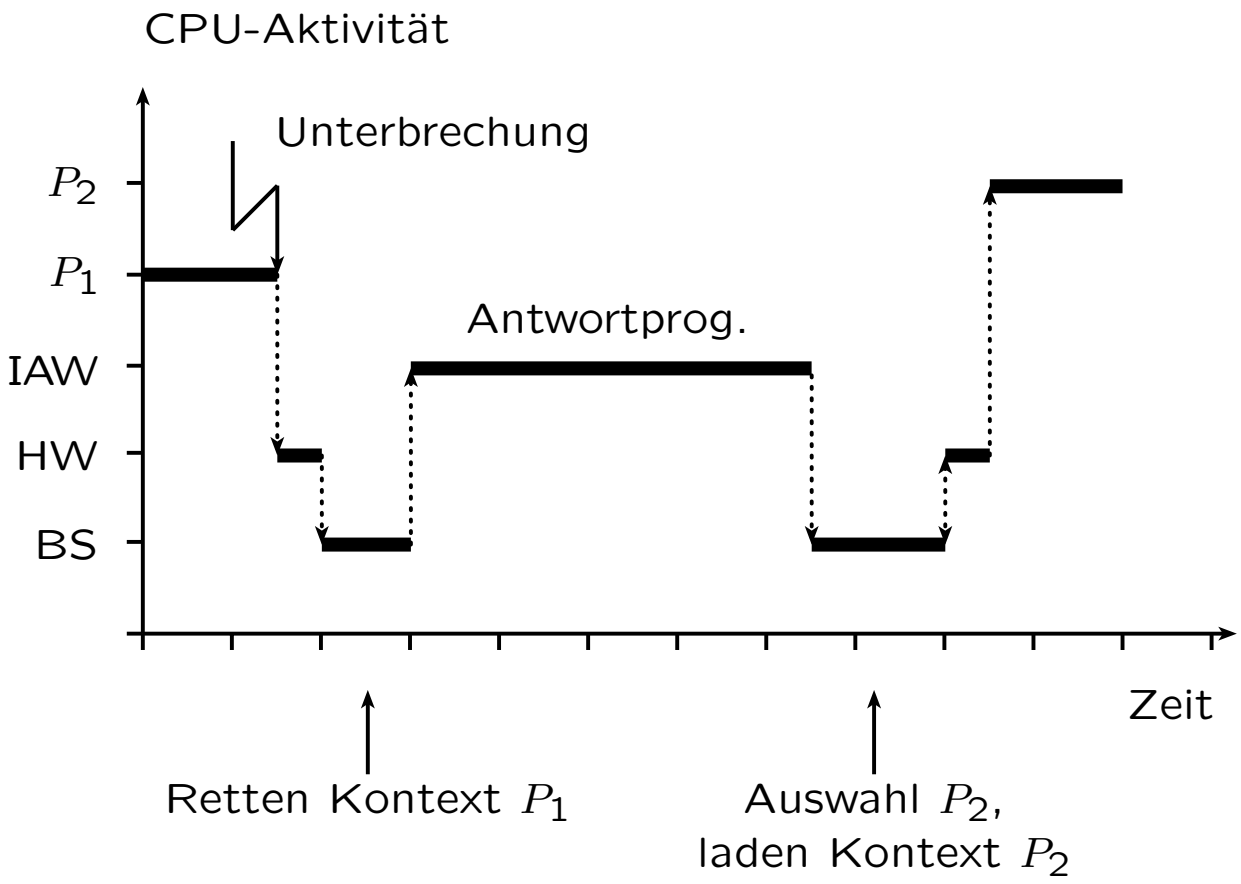
```
TASK master_1;
   <Deklarationen>
   <Initialisierung von Daten und Geraeten>
   ...
   ACTIVATE P1;
   AT 7:00 ACTIVATE P2;
   AT 12:00 ALL 1 min ACTIVATE P3
      UNTIL 16:30;
   ...
   LOOP
      RECEIVE botschaft FROM ANY;
      CASE botschaft.typ OF
         b1: <Fall Prozeß Pi beendet> ;
         b2: <Fall Neuplanung erforderlich> ;
         b3: <Fall Fehler in Pi aufgetreten> ;
      END CASE;
   END LOOP;
END master_1;
TASK Pi;
   <Deklarationen>
   ...
   SEND <botschaft>  TO master_1;
     /* z.B. Botschaft "b1" */
END Pi;
```

⋆ Beispiel zu Variante 2:

```
TASK master_2;
   <Deklarationen>
   <Initialisierung von Daten und Geraeten>
   ...
   SEND <starte>  TO P1;
   SEND <starte>  TO P2;
   ...
   LOOP
      RECEIVE botschaft FROM P1 OR P2;
      CASE botschaft.typ OF
         b1: <Fall Ergebnis von Pi> ;
         b2: <Fall Fehler in Pi> ;
      END CASE;
   END LOOP;
END master_2;
TASK Pi;
   <Deklarationen>
   <Initialisierungen>
   LOOP
      RECEIVE <starte>  FROM ANY;
      Auftrag aus starte ausführen;
        /* jetzt Antwort an Auftraggeber senden */
      SEND <ergebnis>  TO <client> ;
   END LOOP;
END Pi;
```

9.5 Unterbrechungs–Antwortprogramme

- Unterbrechungssignale (Interrupts)
  - ⋆ von Geräten, Uhr, Signalgebern usw.
  - ⋆ haben Unterbrechungspriorität
  - ⋆ können Rechenprozesse niedererer Priorität unterbrechen
- Unterbrechungsvektor
  - ⋆ im Arbeitsspeicher des BS
  - ⋆ für jede Unterbrechung dort Anfangsadresse des Antwortprogramms und Prozessorstatuswort
- Unterbrechung
  - ⋆ Kontext des laufenden Prozesses P_i retten
  - ⋆ Anfangsadresse Antwortprogramm und Prozessorstatuswort aus Unterbrechungsvektor holen
  - ⋆ Unterbrechungs–Antwortprogramm aufrufen
  - ⋆ unterbrechbar durch höherpriore Abläufe
  - ⋆ nach Rückkehr Kontext von P_i restaurieren
  - ⋆ P_i oder höherprioren Prozeß P_j fortsetzen
  - ⋆ unterbrechbar durch höherpriore Abläufe


CPU-Aktivität
Unterbrechung
P_2
P_1
IAW
HW
BS
Antwortprog.
Zeit
Retten Kontext P_1
Auswahl P_2,
laden Kontext P_2

- Fünf Klassen von Unterbrechnungsantwortprogrammen

  a Mini-UAP
  keine Interaktion mit Betriebssystem
  nur Zugriff auf E/A-Register

  b UAP mit nichtblockierendem Systemaufruf
  geeigneter Mechanismus zum Umschalten in Betriebssystemadreßraum (`enter kernel`)
  Dienst nicht an jeder Stelle ausführbar

  c Kombination aus a und b
  In der Praxis oft mit kurzen ununterbrechbaren UAPs und Systemdiensten

  d wie c, aber mit Pufferung der Systemaufrufe, falls BS gerade ununterbrechbar

  e Maximallösung
  direkter Start einer Task
  sehr mächtig, aber langsam

- Zu Unterbrechungen gehören auch "Software–Interrupts"
  - ⋆ (Ausnahmesignale, traps, exceptions)
  - ⋆ durch Anweisungen des BS oder der Benutzerprogramme ausgelöst

- Sprachliche Mittel zur Anbindung von Unterbrechungs–Antwortprogrammen
  - ⋆ Taskstart:

    WHEN <Unterbrechungssignals> DO
      ACTIVATE <Taskname> ;
  - ⋆ Antwortprozedur:

    ON <Unterbrechungssignals> DO
      <Statement-Sequenz>
      <setze unterbrochenen Prozeß fort >
    END ON
  - ⋆ <Name des Unterbrechungssignals> muß Compiler/Binder auf die Hardware–Interrupt–Nummer abbilden
  - ⋆ Adresse des Antwortprogramms wird dynamisch im Unterbrechungs–Vektor eingetragen
  - ⋆ Antwortprogramm muß bei dieser Lösung statisch im Speicher bleiben.
  - ⋆ Eine saubere sprachliche Einbindung ist schwierig (Gültigkeitsbereiche, Rechte), daher existieren viele verschiedene Lösungen

9.6 UNIX für Echtzeitaufgaben?

- Warum UNIX
  - ⋆ Bei Cross–Entwicklungen möchte man nicht zwei Betriebssysteme kennen müssen
  - ⋆ Problem 1: UNIX nicht echtzeitfähig
  - ⋆ Problem 2: Bei kleinen Echtzeitanwendungen (embedded systems) fehlt notwendige HW (Platte, MMU)
- 1969 UNIX als Mehrbenutzersystem bei BELL durch Ken Thompson und Dennis Ritchie entwickelt
- Später viele Versionen, u.a. System V
- SVID: UNIX SYSTEM V Interface Definition
- seit 1984 Standardisierungen der Schnittstellen
  - ⋆ X/OPEN
    einheitliche Systemumgebung auf Basis SVID
  - ⋆ IEEE 1003
    Schnittstellen auf Basis SVID, u.a.
    - ◦ POSIX (Portable Operating System UNIX)
    - ◦ Realtime facilities (IEEE 1003.4,4A)

- POSIX Real–Time Extensions (IEEE 1003.4)
  - ⋆ Prioritäten für Prozesse
  - ⋆ Präemption
  - ⋆ schneller Kontextwechsel
  - ⋆ Speicherfixierung aktiver Prozesse
  - ⋆ asynchrone E/A (Direkt–DMA von Gerät zu Prozeß)
  - ⋆ feineres Zeitraster (Uhren, Wecker)
  - ⋆ "Contiguous"–Dateien mit festen Plattenbereichen
  - ⋆ einfache Blockverwaltung bei Plattenspeicher
  - ⋆ Hilfsmittel für gemeinsame Datenbereiche
- Ausbauklassen (IEEE 1003.13)
  AEP (application environment profiles)

(1) Minimal real–time system for embedded systems (realtime kernel, threads)

(2) Real–time controller system (E/A, Dateien im Speicher, Ausnahmen)

(3) Dedicated real–time system (Prozesse, evtl. MMU)

(4) Multi–purpose real–time system (Vollausbau, Massenspeicher, Netz, FTAM)

- Was fehlt UNIX für Echtzeitverhalten?
  - ⋆ Zeit nicht deterministisch projektierbar
    - ◦ System zu langsam (lange ununterbrechbare Phasen)
    - ◦ 100 – 1000 msec Reaktionszeit auch bei nur einem Benutzerprozeß
    - ◦ großer Prozeßkontext
    - ◦ keine "echten" Prioritäten
    - ◦ Zeitscheibentechnik
    - ◦ keine hauptspeicherresidenten Tasks
    - ◦ Kern nicht reentrant
  - ⋆ E/A- Zugang nur über vorhandene Treiber und Systempuffer
  - ⋆ Dateistruktur zu allgemein
  - ⋆ viele Kopiervorgänge zwischen Puffern
  - ⋆ keine geeignete Prozeßkommunikation:
    - ◦ Pipes (mit Kopieren verbunden),
    - ◦ Signale (unsicher);
    - ◦ Semaphor zu langsam implementiert
  - ⋆ SVID–Dienste meist zu allgemein und deshalb langsam

- Fünf Vorgehensweisen zur Abhilfe:

(1) Modifikation des Kerns, u.a.

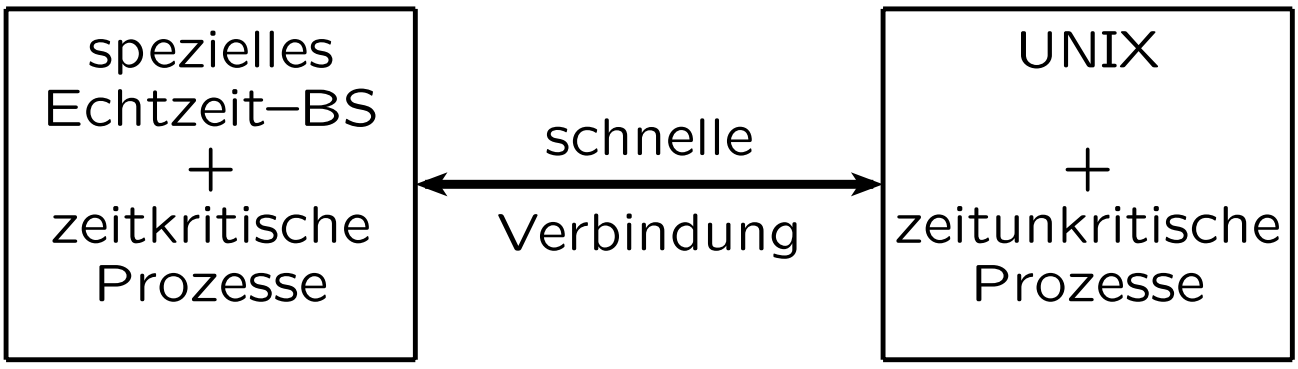
  - ⋆ Prioritäten
  - ⋆ unterbrechbare Stellen im Kern

(2) Implementierung eines echtzeitfähigen Minikernels
Linux-Betriebsystemkern nur eine Echtzeittask mit der niedrigsten Priorität

(3) neuer Kern bei gleichen Aufrufen, z.B. LynxOS

(4) SVID oder POSIX auf geeignetem Echtzeit–Betriebssystem aufsetzen

(5) heterogenes Mehrrechnersystem



9.7 RT-Linux

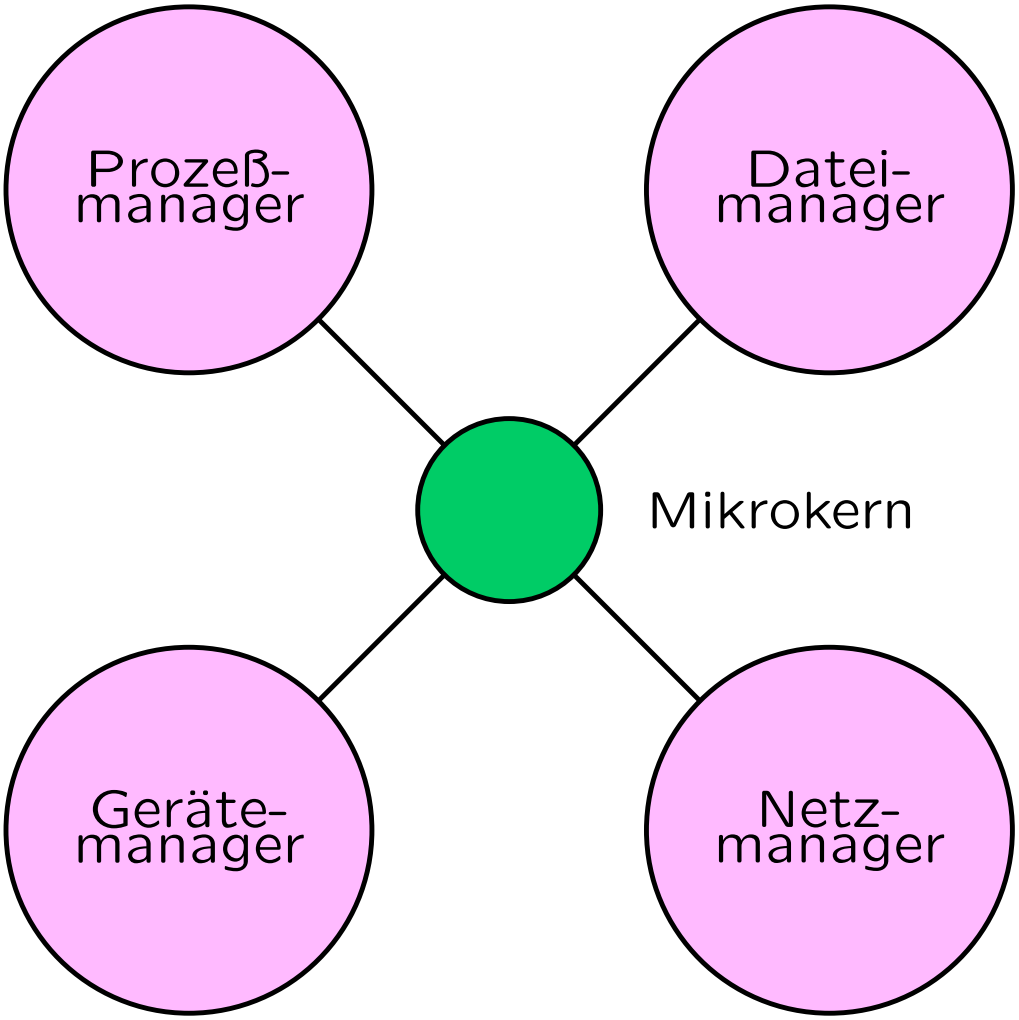
Grundidee:

- Implementierung eines Echtzeitkerns
- Linux-Kern läuft als Task mit kleinster Priorität
- Kommunikation der Echtzeittasks mit Linux Prozessen über FIFO-Puffer

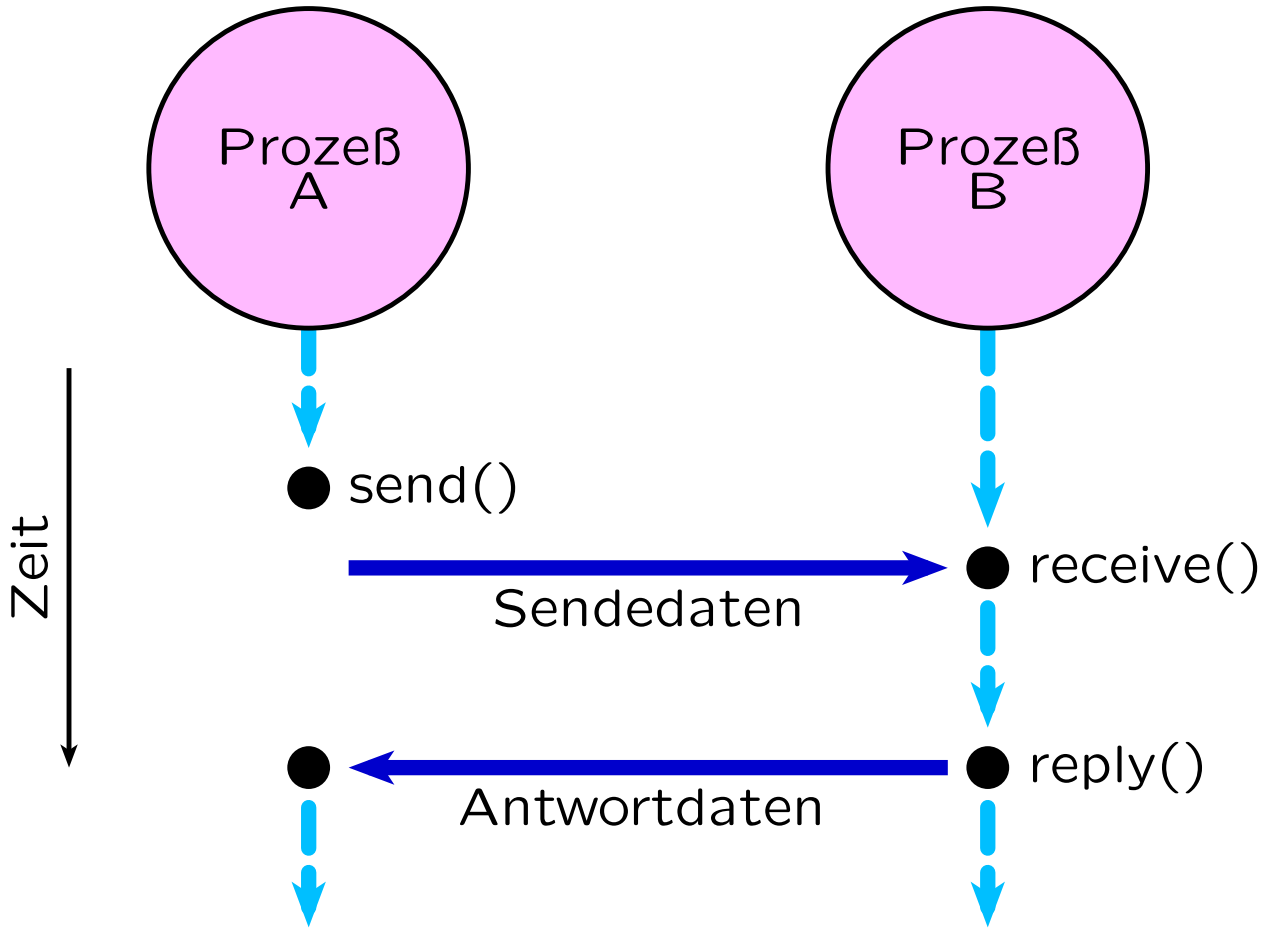
Unter
`http://www.realtimelinuxfoundation.org/`
`http://www.fsmlabs.com/`
`http://www.aero.polimi.it/rtai/`
finden sich detaillierte Beschreibungen zur Architektur von RT-Linux.

9.8 QNX realtime OS

- besteht aus Mikrokern, System- und Anwenderprozessen
- basiert auf Standard-PC-Hardware
- frei verfügbar unter `http://get.qnx.de/`
- Systemprozesse (außerhalb Mikrokern)
  - ⋆ Prozeßmanager
  - ⋆ Dateimanager
  - ⋆ Gerätemanager
  - ⋆ Netzmanager



- Mikrokern
  - ⋆ klein (auch für eingebettete System im ROM)
  - ⋆ skalierbar
  - ⋆ schnell (Kontextwechsel einige Mikrosekunden)
  - ⋆ kleiner BS-Overhead
  - ⋆ zuverlässig
- nachrichtenbasierte Interprozeßkommunikation



Anmerkung: receive nach FIFO oder nach Priorität

- Multitasking
- Multithreading
- präemptive RK-Vergabe nach Prioritäten
- Aufgaben Mikrokern
  - ⋆ Erzeugung Prozesse (soweit nicht im Prozeßmanager)
  - ⋆ Speichermanagement
  - ⋆ Zeit-, Wecker-, Uhrenverwaltung
  - ⋆ Interprozeßkommunikation
  - ⋆ Behandlung Unterbrechungen und Signale
  - ⋆ RK-Vergabe
- transparentes verteiltes System
  - ⋆ starten von Prozessen an beliebigem Ort des Netzes
  - ⋆ neue Prozesse erben Umgebung des Vaters, u.a.:
    - ◦ geöffnete Dateien
    - ◦ aktueller Pfad
    - ◦ Dateideskriptoren
    - ◦ Benutzer-ID

- POSIX-Eigenschaften, u.a.:
  - ⋆ POSIX.4 Uhren und Wecker
    - ◦ mehrere Wecker je Prozeß
    - ◦ Auflösung in Nanosekunden
    - ◦ flexible Weckersteuerung: Wecker arbeiten synchron oder asynchron; einmalig oder periodisch wiederholend
  - ⋆ voll geschachtelte Unterbrechungen
  - ⋆ dynamisch zuordenbare Unterbrechungsbehandlungen
  - ⋆ flexible Primitive für einen gemeinsamen Speicher
  - ⋆ eingebaute Testprimitive zur Fehlerlokalisierung irgendwo im Netz
  - ⋆ durch Benutzer konfigurierbare Grenzen für Ressourcen
  - ⋆ netzwerkuniverselle Namen

- ⋆ POSIX.4 Realzeit Scheduling
  - ◦ 32 Prioritäten
  - ◦ präemptiv, RK-Vergabe gemäß Priorität
  - ◦ Strategien (prozeßspezifisch wählbar)
    - ▷ grundsätzlich bei allen Strategien Präemption durch höherprioren Prozeß überlagert
    - ▷ FIFO (first in first out)
    - ▷ Zeitscheiben (round robin) Zeitscheibe (50 ms)
    - ▷ adaptive RK-Zuteilung Zeitscheibenverfahren; am Ende einer Zeitscheibe wird die Priorität eines Prozesses um 1 erniedrigt, falls sie nicht schon erniedrigt wurde; blockiert der Prozeß erhält er seine urpsrüngliche Priorität wieder
- ⋆ Server können ihre Priorität abhängig von ihren Aufträgen wechseln

- Dateisysteme
  - ⋆ mehrere verschiedene Dateisysteme können zur gleichen Zeit laufen
  - ⋆ POSIX-Dateisystem
  - ⋆ DOS-Dateisystem
  - ⋆ Netz-Dateisystem SMB (Server Message Block) für Windows NT, Windows 95, Windows for Workgroups, LAN Manager
  - ⋆ Netz-Dateisystem NFS
  - ⋆ CD-ROM-Dateisystem
  - ⋆ elementares Block-Dateisystem
- Fehlertoleranz und Flexibilität des Netzes
  - ⋆ dynamisches Zufügen, Entfernen, Anhalten und Fortsetzen von Knoten ohne explizite Rekonfiguration durch den Benutzer
  - ⋆ automatische Erkennung von Netzänderungen (Hinzufügen, Entfernen von LANs)
- Konfigurierung von QNX gemäß Bedarf der Anwendung

9.9 LynxOS

- Hard Real-Time Performance
- POSIX 1003 konform
- 256 Prioritäten für kernel threads
- präemptiver EZ-BS-Kern
- schneller Kontextwechsel, deterministische Blockierungszeiten
- Priority inheritance
- Semaphore, mutex, events, message queues
- clocks und timer
- UNIX-Dateisystem und Netzkommunikation
- skalierbar

Unter
`http://www.lynuxworks.com/products/whatislos.html`
finden sich detaillierte Beschreibungen zur Architektur und zur Leistungsfähigkeit von LynxOS

9.10 VxWorks 5.4

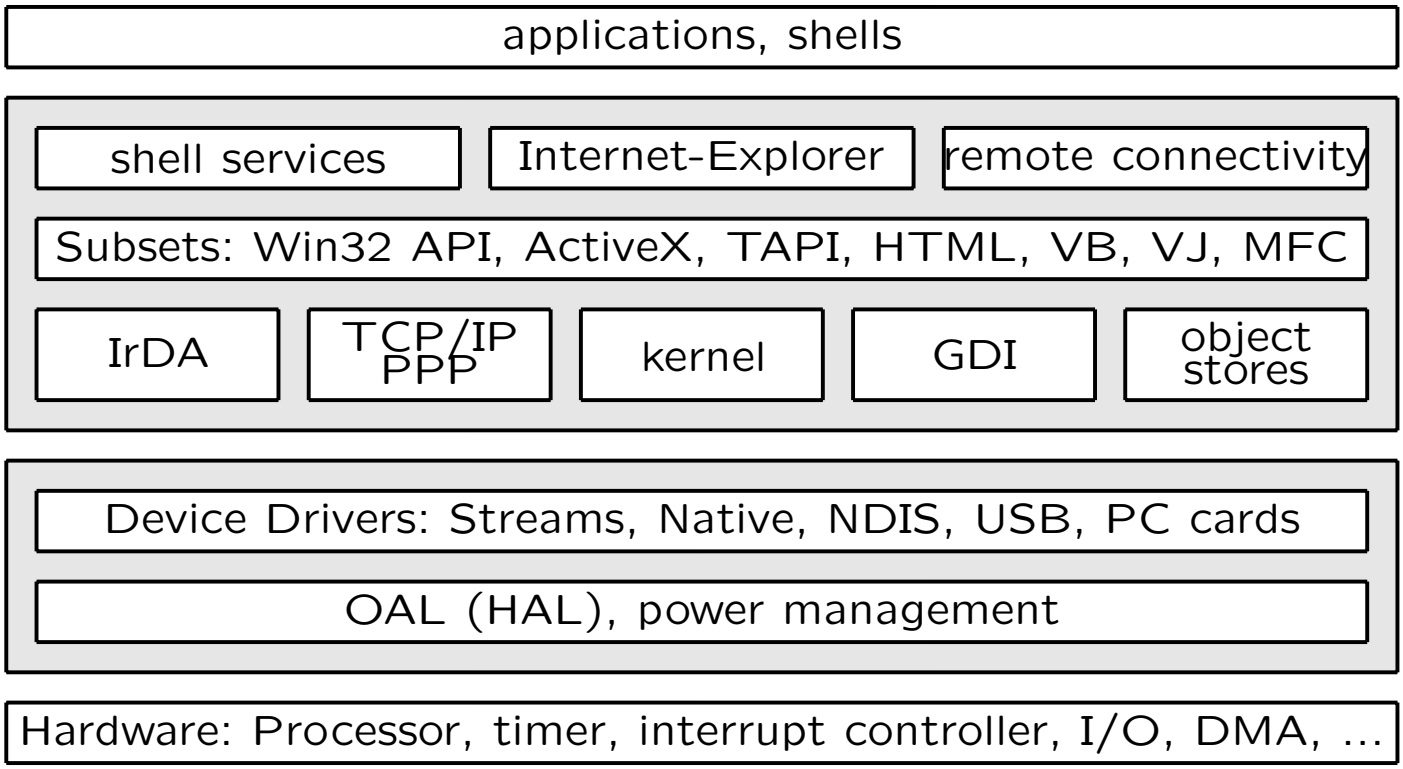
- siehe Folien zum Vortrag von
  Herrn Hager von der Firma Wind River
- Unter

  `http://www.windriver.com/products/family/ide.html`

  finden sich detaillierte Beschreibungen zur Architektur von VxWorks und der Entwicklungsumgebung Tornado

9.11 Windows-CE

- modular, schichtenorientiert
- Baukastensystem
- skalierbar (ca. 200 KB bis 8MB)
- versch. Hardware-Plattformen
- 32 bit
- MS-Windows-kompatibel (Teilmenge Win32 API)
- teilweise NT-Schnittstellen
- viele Kommunikationsprotokolle
- deterministisch
- derzeit noch "weiches" Realzeitverhalten
- 32 (Schwergewichts-) Prozesse
- dazu threads in den einzelnen Prozessen
- virtuelles Speichermanagement
- COM (Component Object Model) und ActiveX
- Spiegelung Dateisystem möglich
- integriertes Energie-Management
- erster Einsatz 1996 (handheld computer)



Schichten und Komponenten Windows CE