C Betriebssystemarchitekturen

C.1 Entwicklung der Betriebssystemstrukturierung

- Monolithische Kerne
- Minimalkerne
- Objektbasierte Systeme & Sandboxing

C.2 Monolithische Betriebssystemkerne

- Alle Betriebssystemkomponenten zu einem Kern zusammengefaßt
- Ausführung generell im privilegierten Modus

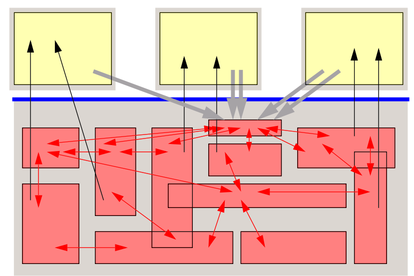
1 Vorteile

+ Effiziente Kommunikation zwischen den Komponenten des Kerns durch Prozeduraufrufe
+ Direkter Zugriff auf alle Datenstrukturen des Kerns jederzeit möglich
+ Privilegierte Befehle jederzeit aufrufbar

2 Nachteile

- Mechanismen strikt vorgegeben
  - ◆ individuelle Anpassung an Anwendungen nicht möglich
  - ◆ Erweiterung oder Reduzierung in modernen Systemen teilweise durch dynamisch ladbare/entfernbare Module (vor allem Treiber, Dateisysteme)
- Strukturierung meist schlecht
  → änderungsunfreundlich und fehleranfällig
- Kein Schutz gegenüber Kernkomponenten möglich
  (kritisch z. B. bei zugekauften Treibern!)
- Niedriges Abstraktionsniveau
  Programmierung "direkt auf der nackten Hardware"

3 Struktur



billige, ungeschützte Interaktion
teuere, geschützte Interaktion

Adreßraum
Anwendung
Betriebssystemfunktion

C.3 Minimalkerne

- Auslagerung von Betriebssystemkomponenten
  - ➔ Plazierung der Komponenten in eigenen Adreßräumen
  - ➔ Ausführung im Benutzermodus
- Reduktion des Kerns auf Basisfunktionalität und die Mechanismen, für die privilegierter Modus notwendig ist

1 Vorteile

\+ Bessere Modularisierung

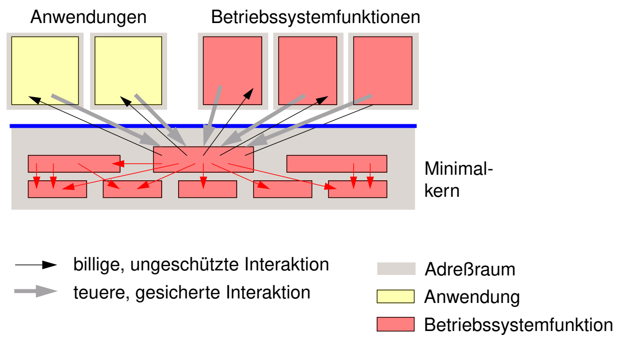
\+ Ausgelagerte Teile leichter individuell zu gestalten
  - ➔ zusätzlich angepaßte Module erzeugbar
  - ➔ nicht benötigte Module können einfach weggelassen werden

\+ Ausgelagerte Teile werden nicht privilegiert ausgeführt → + Sicherheit

2 Nachteile

– Kommunikation zwischen Modulen teuer
(RPC, 20- bis 70-facher Aufwand eines Prozeduraufrufs)
  - ➔ Module werden aus Effizienzgründen möglichst groß gehalten
  - ➔ innerhalb der Module Strukturierung beliebig
  - ➔ aus Effizienzgründen verbleiben Komponenten im Minimalkern, die nicht dorthin gehören

3 Struktur



C.4 Objektbasierte Systeme

- Fortführung des Wegs ’monolithischer Kern → Minimalkern’
- Weitere Modularisierung nur möglich, wenn Inter-Modul-Kommunikationskosten reduziert werden
  - ➔ Adreßraumkapselung muß aufgegeben werden
  - ➔ Verlust an Sicherheit
  - ➔ Sicherheit muß anders gewährleistet werden

➥ Sicherung von Modulgrenzen
- ◆ auf Sprachebene
- ◆ beim Laden von Software
- ◆ durch spezielle Laufzeit-Überprüfung

1 Sandboxing

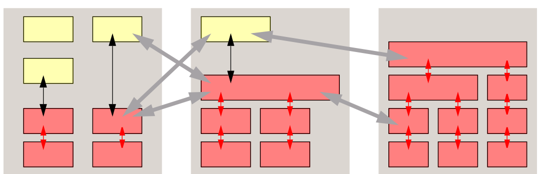
= Garantieren von Modulgrenzen auf Softwareebene

- Sprachunterstützung
  - ➔ objektorientierte bzw. objektbasierte Programmierung
- Kontrolle beim Laden von Software
  - ◆ Byte-Code-Verifier (z. B. bei Java Byte Code)
  - ◆ Einfügen von prüfendem Code (bei Maschinencode)
- "Betriebssystem-artige" Komponenten prüfen zusätzlich, ob Anforderungen eines Moduls zulässig sind
- Wenn Sicherheit auf Sprachebene oder beim Laden garantiert wird, kann auf Adreßraumgrenzen verzichtet werden, sonst nicht!
  - ➔ gleiche Modul-Interaktion adreßraum-lokal und adreßraum-übergreifend wünschenswert
  - ◆ nicht-vertrauenswürdige Module werden mit MMU-Hilfe gekapselt

2 Vorteile

\+ Strukturierungsmöglichkeiten von Systemen mit ausgelagerten Komponenten mit Effizienz eines monolithischen Kerns kombinierbar

\+ Individuelle Anpaßbarkeit wie bei einem Minimalkern möglich
→ durch feinere Granularität der Module jedoch weit flexibler



—► billige, durch Objektkapselung geschützte Interaktion

⇨ teuere, durch Adreßraumgrenze geschützte Interaktion

Adreßraum
Anwendungsobjekte
Betriebssystemobjekte

C.5 Betriebssysteme für Embedded Systems

- Grundsätzlicher Unterschied zu "general-purpose"-Systemen
  - ◆ Anforderungen sind vor Systemstart bekannt
  - ◆ auszuführende Applikationen sind bekannt
  - ◆ keine oder nur wenig Dynamik im laufenden System
  - ◆ kein Schutz gegen "böswillige" Programme notwendig
    - ➥ keine Adreßraumgrenzen
- Realisierung
  - ◆ meist Bibliothek mit Betriebssystemfunktionen
    - ➤ Betriebssystemfunktionen werden statisch mit Anwendung gebunden
    - ➤ Betriebssystemfunktionen als "dynamic link library"
- Beispiele
  - ◆ Windows CE (breites Anwendungsspektrum)
  - ◆ OSEK-Architektur (Zielanwendung: KFZ-Elektronik)

C.6 Anforderungen an zukünftige Betriebssysteme

- Betriebssysteme müssen an die Anforderungen von Anwendungen anpaßbar sein
  - – Effizienz
  - – Schutzanforderungen
  - – Zugriffs- oder Cache-Strategien
  - ➔ Anwendungen dürfen nicht gezwungen sein, Betriebssystemmechanismen zu umgehen
- Höheres Abstraktionsniveau bei der Programmierung von Betriebssystemkomponenten
  - ➔ Anpassung, Erweiterung und Wartung von Betriebssystemen einfacher
- Betriebssysteme für verteilte Systeme, Mehrprozessorsysteme, Parallelrechner