

# K Verteilte Betriebssysteme

---



## Verteilte Betriebssysteme

© 2001-2004, F. Hauck, P. Schulthess, Vert. Sys., Univ. Ulm  
Reproduktion oder Verwendung dieser Unterlage bedarf in jedem Fall der Zustimmung des Autors.

[2003w-VBS-K-DOS.fm, 2004-02-11 09:02]

K - 1

## 1 Eigenschaften

---

- Verteiltes Betriebssystem
  - ◆ gebildet durch zusammenarbeitende lokale Betriebssysteminstanzen
  - ◆ Programmiermodell für verteiltes Programmieren
  - ◆ Umgebung für verteilte Programme
    - Ausführungsplattform
    - Dienste: z.B. Namensdienst
  - ◆ Integration in das Betriebssystem:
    - Kommunikationsmechanismen für Verteilungseinheiten
    - Identifikation der Verteilungseinheiten



## Verteilte Betriebssysteme

© 2001-2004, F. Hauck, P. Schulthess, Vert. Sys., Univ. Ulm  
Reproduktion oder Verwendung dieser Unterlage bedarf in jedem Fall der Zustimmung des Autors.

[2003w-VBS-K-DOS.fm, 2004-02-11 09:02]

K - 2

## 2 Amoeba

---

- Vrije Universiteit Amsterdam (VU),  
Centrum voor Wiskunde en Informatika (CWI)
  - ◆ Andrew Tanenbaum, Sape Mullender et al. 198x

### 2.1 Programmiermodell

---

- Objekte mit Methoden
  - ◆ eingebettet in verteilte und migrierbare Serverprozesse
    - multi-threaded
  - ◆ orts- und verteilungstransparente Methodenaufrufe
    - schnelle Amoeba-RPC-Implementierung
    - Lokalisierung des Serverprozesses als Teil des Protokolls
      - zunächst nur für lokale Cluster geeignet
      - über Proxy-Prozesse für große Netze nutzbar



### 2.1 Programmiermodell (3)

---

- Amoeba-Interface-Language (AIL)
  - ◆ zur Erzeugung von Client- und Server-Stubs
- Betriebssystemschnittstelle und Dienstschnittstellen
  - ◆ weitgehend als Amoeba-Objekte definiert
- Objektreferenzen
  - ◆ als Capability realisiert
    - Referenz plus Zugriffsrecht
  - ◆ durch kryptographische Methoden vor Verfälschung gesichert
  - ◆ Rechte können vor Weitergabe beschränkt werden



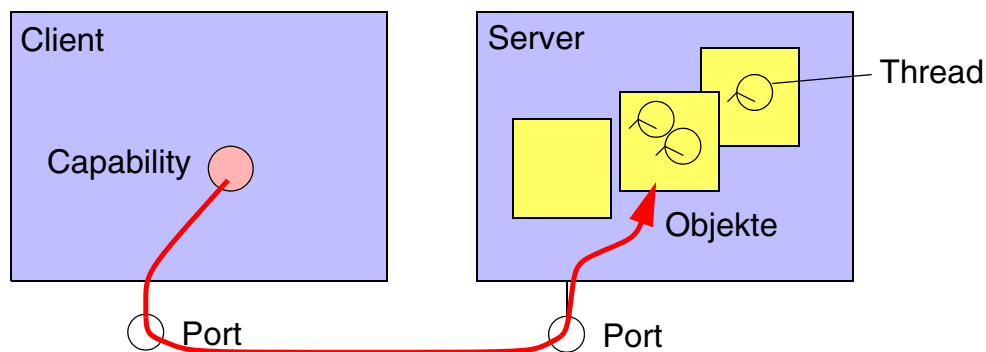
## 2.2 Prozessverwaltung

- Ressourcen eines Prozesses
  - ◆ Ports
    - Endpunkte für Clients zum Aufruf von Prozessen
  - ◆ Speichersegmente
  - ◆ Threads
- Prozessmigration
  - ◆ Einfrieren eines Prozesses und Übertragung aller Ressourcen
    - nur zwischen gleichen Rechnerarchitekturen
- Entferntes Debugging beliebiger Prozesse
  - ◆ falls Prozess-Capability entsprechende Rechte hat



## 2.2 Prozessverwaltung (2)

- Überblick



## 2.3 Dateisystem

---

- Namensdienst für beliebige Objekte
  - ◆ Verzeichnisstruktur (Baum)
  - ◆ Abbildung von Namen auf Capabilities
    - differenzierte Abbildung je nach Benutzer
  
- Bullet-File-Server
  - ◆ Dateiobjekte
  - ◆ nur einmal beschreibbar
    - einfache Konsistenzsicherung
  - ◆ Umtragen der Capabilities im Namensdienst bei mehrfachem oder nebenläufigem Schreiben



## 2.4 Unix-Emulation

---

- Unix-Systemschnittstelle
  - ◆ Einsatz von Standard-Unix-Werkzeugen möglich
  
- X-Window-Implementierung
  - ◆ TCP/IP-basiert und Amoeba-RPC-basiert

## 2.5 Einsatz

---

- Bis Mitte der Neunziger Jahre im Einsatz in Forschung und Lehre
  - ◆ komplexes Verteiltes Betriebssystem in vollständiger Eigenentwicklung



## 3 Plurix

---

- Abteilung Verteilte Systeme, Prof. Peter Schulthess et al.
  - ◆ seit Ende der Neunziger Jahren
- Zielsetzung
  - ◆ schlankes Verteiltes Betriebssystem mit gemeinsamem verteilten Speicher
    - BS-Code im Bereich etwa 200 KB
  - ◆ gemeinsamer Adressraum für alle Knoten eines Clusters
    - keine Kontextwechsel
    - kein RPC notwendig
    - keine Serialisierung von Daten notwendig
  - ◆ Verzicht auf heterogene Rechnerarchitekturen
  - ◆ direkte Hardwareprogrammierung
    - echtes Betriebssystem



## 3 Plurix (2)

---

- Zielsetzung
  - ◆ Einsatzgebiete
    - Hochleistungsrechnen
      - z.B. Ray-Tracing
    - Allgemeine Datenverarbeitungsaufgaben
    - Multimedia-Telekooperation
    - Multiplayer-Spiele
    - Virtuelle Welten



## 3.1 Programmiermodell

---

- Java-Objekte
  - ◆ eigener Java-Compiler
  - ◆ Java-Programmiermodell im verteilten System
    - Objekte liegen im DSM (Heap)
  - ◆ keine explizite Verteilung notwendig
    - Objekte „wandern“ an den Ort des Zugriffs
  
- Transaktionen
  - ◆ alle Aktionen laufen als Teil einer Transaktion
  - ◆ optimistisches Implementierungsverfahren
    - Voraussetzung: kurze Transaktionen bzw. unwahrscheinliche Konflikte



## 3.1 Programmiermodell (2)

---

- Transaktionen (fortges.)
  - ◆ Serialisierung der Auswertungsphase
    - Token wird im System verschickt
    - nur Station mit Token kann Transaktion auswerten
  - ◆ Vorwärtsinvalidierung
    - Vergleich des Write-Sets der auszuwertenden Transaktion mit den Read-Sets aller noch laufenden Transaktionen
    - Abbruch der noch laufenden Transaktionen im Konfliktfall

