Verteilte Betriebssysteme

3. Kapitel
Fallstudien Verteilter Betriebssysteme

Matthias Werner
Professur Betriebssysteme

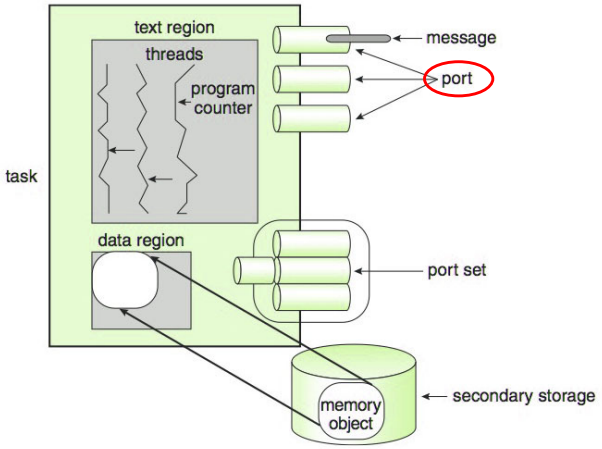
3.1 Motivation

- Bevor wir einzelne Aspekte diskutiert, betrachten wir einige **reale** Betriebssysteme
- Betriebssysteme für verteilte Systeme müssen verschiedene Probleme lösen
  - Bisher keine einheitliche Problemsicht ➡ unterschiedliche Konzepte und Abstraktionen
  - **Sehr** heterogen
  - Noch kaum etablierte *best practices* oder gar Standards, erst in letzter Zeit Bemühungen im Rahmen des Cloud-Computing
- Konzentrieren uns hier auf Abstraktionen
- Unterschiedliche Abstraktionen (Modelle) stellen unterschiedliche Anforderungen an Implementierung

3.2 Mach – Tore als Stellvertreter

- Entwickelt 1983–86 von der Carnegie-Mellon-University (CMU) in Pittsburgh
- Vorgänger: Accent (1981, CMU)
- Ursprüngliches Ziel: Moderne Neuimplementierung von UNIX BSD 4.3
- BS-Kern Mach 3.0 verfügbar für die meisten PC- und Workstation-Prozessoren
- Grundlage von OSF/1, dem Unix-Standard der Open System Foundation
- Kernel für:
  - NeXTSTEP / OPENSTEP - Rhapsody
  - XNU (Basis von Darwin, was wiederum die Basis von MacOS X ist)
  - MkLinux (Power Macintosh)
  - GNU Hurd (in Form von GNU Mach)
  - Tru64 (OSF/1)

Mach-Objekte



Ports

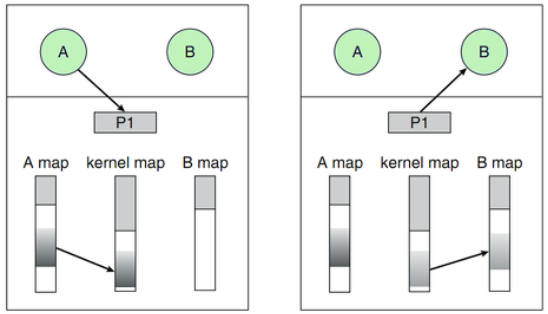
- **Ports** spielen Schlüsselrolle in Mach
  - Typisch: ein Server verwaltet viele Ports
  - **Betriebsmittel** werden mit Ports identifiziert und ihre Benutzung wird durch eine Nachricht an den zugehörigen Port angefordert
- Zugriff auf Betriebsmittel wird über Ports geregelt ➡ **Recht** auf Benutzung eines Betriebsmittels wird über ein Recht am zugehörigen Port realisiert
  - Die Besitzer der Rechte sind Tasks
  - Rechte können in Nachrichten verschickt werden
  - Folgende Rechte sind vorgesehen:
    - send right
    - send-once right
    - receive right
    - bootstrap port right

Ports (Forts.)

- Ports unterstützen n:1-Kommunikation
- Für jeden Port gibt es zu einem Zeitpunkt **genau einen** Empfänger (Besitzer des receive rights), aber beliebig viele Sender (Besitzer eines send rights)
- Ports bestehen im Wesentlichen aus einer Nachrichtenwarteschlange, deren Größe vom Port-Besitzer dynamisch eingestellt werden kann (Flusskontrolle)
  - Senden an einen vollen Port führt zur Blockierung des sendenden Threads
  - Lediglich Sende-Operationen mit send-once right sind auch bei voller Warteschlange erfolgreich
- Existiert kein receive right für einen Port, so werden Sende-Operationen mit einer Fehlermeldung abgewiesen
- Ebenso kann ein Prozess den Kern beauftragen, ihm mitzuteilen, wenn keine send rights mehr existieren

Ports (Forts.)

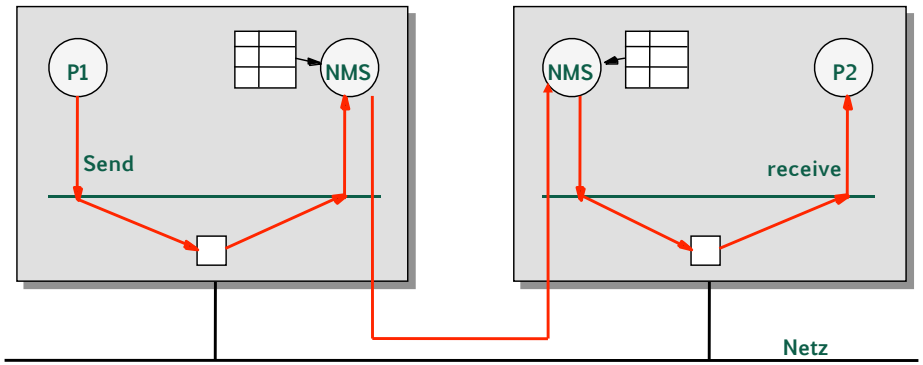
- Der Kern führt Buch über die Zahl der ausgegebenen Rechte für einen Port
  - Verwaltung erfolgt im Kernel
  - Portname (Integer) ist lokal bezüglich des Prozesses
  - Kernel isoliert damit Prozesse ➡ Sicherheitsfeature



Entfernte Kommunikation

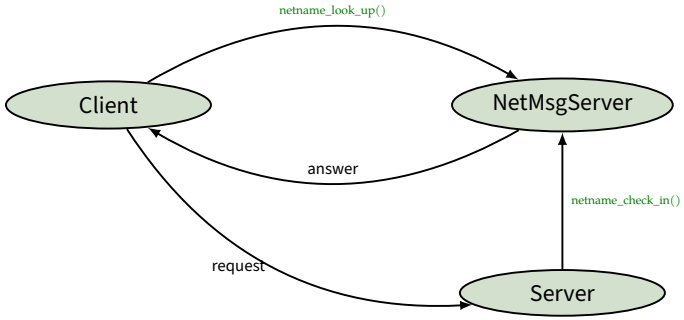
- Mach ist zunächst kein verteiltes System
- **Aber:** Das Konzept „Port“ als zentrale Abstraktion macht es leicht, verteilte Betriebssysteme auf der Grundlage von Mach zu konstruieren
- Kommunikation zu anderen Knoten ist Aufgabe einer speziellen Komponente außerhalb des Kerns: **Net Message Server** (NetMsgServer, NMS)
- NMS-Prozess realisiert die typischen Schichten eines Protokollstapels
- Er enthält u.a.:
  - Tabellen zur Umsetzung lokaler Portnummern in globale
  - Funktionen zur Wandlung der Datenrepräsentation
  - Authentisierungsmechanismen
  - Verschlüsselungsverfahren

Entfernte Kommunikation (Forts.)



Arbeiten mit dem NetMsgServer

- Dienste registrieren sich beim NetMsgServer



- NetMsgServer ist auch Dienst. Wie wird er gefunden?
  ➡ Enviroment-Manager (für lokale Ports) oder Bootstrap-Server (MacOS) mit festem Port

Resümee

Erkenntnis

- Mikrokernel-Ansatz funktioniert bei verteilten und nichtverteilten Systemen
- Zentrales Konzept des Ports ermöglicht „Verstecken“ (Transparenz) von Verteiltheit.

- **Außerdem interessant bei Mach:**
  - Rechteübertragung durch IPC
  - Unterstützung verschiedener Modelle zur Speicherverwaltung, insbesondere Paging
  - **Copy on write** bei neuen Prozessen
  - NORMA (No Remote Memory Access Interprocessor Communication) als In-Kernel-Ersatz für NetMsgServer

Andere Mikrokernel-Ansätze

- **Amoeba**
  - Vrije University/CMCS Amsterdam, 1981
  - Komplettes, Microkernel-basiertes OS mit Prozessor-Pool-Modell, objekt-basiert, verschlüsselte Capabilities
  - Prozessmigration
  - gilt als Referenz für verteilte Betriebssysteme

- **Chorus**
  - INRIA (Frankreich), 1979-87, ursprünglich als europäische Konkurrenz zu Mach
  - Sehr ähnlich zu Mach (konzeptionell), aber kleiner
  - MiX: UNIX-Server, COOL: Chorus Object-Oriented Layer

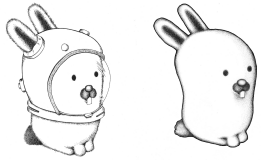
- **V-System**
  - Standford University, 1984
  - Sehr kleiner Kernel
  - Möglichkeiten zur Prozessmigration

3.3 Plan 9 – Alles ist eine Datei

- Vor dem Hintergrund von UNIX vollständig neu entwickelte Rechenumgebung
- Entwickelt in Computing Science Research Center der Lucent Technologies Bell Laboratories (die gleiche Gruppe, die UNIX, C und C++ entwickelt hat, u.a. Rob Pike und Ken Thompson)
- Erste Version erschien 1993 (nur im universitären Bereich verfügbar)
- Offiziell aktuelle Version: Release 4 von 2002
  - Aber ständige Weiterentwicklung durch die „Community“ (und auch noch etwas durch Bell-Labs)
  - Letztes Image vom Mai 2016 (Stand: Oktober 2020)
  - Als Open Source verfügbar, u.a.: `http://www.9legacy.org/download.html`

Trivia

- Name kommt vom Film **Plan 9 From Outer Space**
  - Regie: Ed Wood
  - gewählt zum schlechtesten Film aller Zeiten
- **Glenda** ist für Plan 9 was Tux für Linux ist: das Maskottchen
  - Bezieht sich auf einen anderen Ed-Wood-Film: „Glen or Glenda“



Design-Ansatz

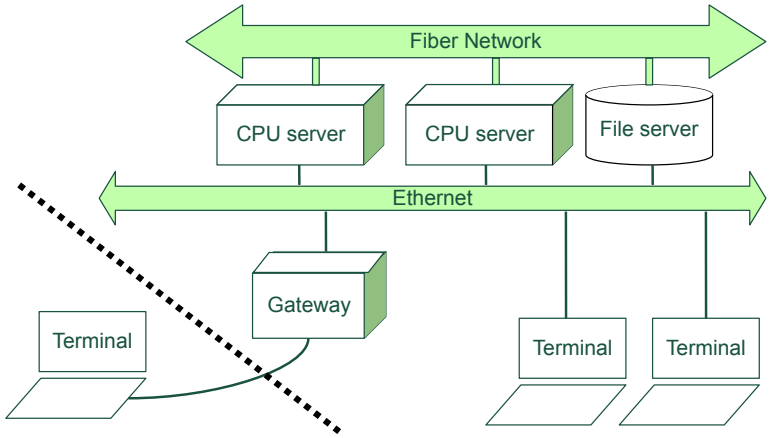
- Unix war ursprünglich kein Netzwerk-Betriebssystem ➡ Brüche im Design

Idee

Ein UNIX, das aus vielen kleinen Systemen besteht, statt einem System, dass aus vielen UNIXen besteht.

- **Plan 9**
  - Ein mächtigeres Multiprozess-System
  - Zentral administrierbar
  - Aufwärtskompatibilität war kein Design-Ziel, aber POSIX-Emulation vorhanden

Typische Plan 9 Umgebung



Verteilungsabstraktion

- UNIX: zwei zentrale Konzepte: Prozess und Datei
- Plan 9: **Alles** ist eine Datei:
  - Gespeicherte Daten und Dateisysteme
  - Fenster
  - Protokolle
  - Netzwerkinterfaces
  - ...
- Umsetzung der Abstraktion
  - Angepasstes **Namensystem** notwendig
  - Jeder Server ist File-Server
  - Implementierung mit einer Art NFS-Protokoll: 9P Protokoll (Version 9P2000 auch unter dem Namen styx)

Beispiele für Dateien

- Eigene Shell terminieren

Terminal
```
; echo kill > /proc/$pid/ctrl
```

- Umgebungvariablen auflisten

Terminal
```
; lc /env
 '*'      cpu      init     planb    sysname
0         cputype  location plumbsrv tabstop
MKFILE    disk     menuitem prompt   terminal
afont     ether0   monitor  rcname   timezone
apid      facedom  mouseport role     user
```

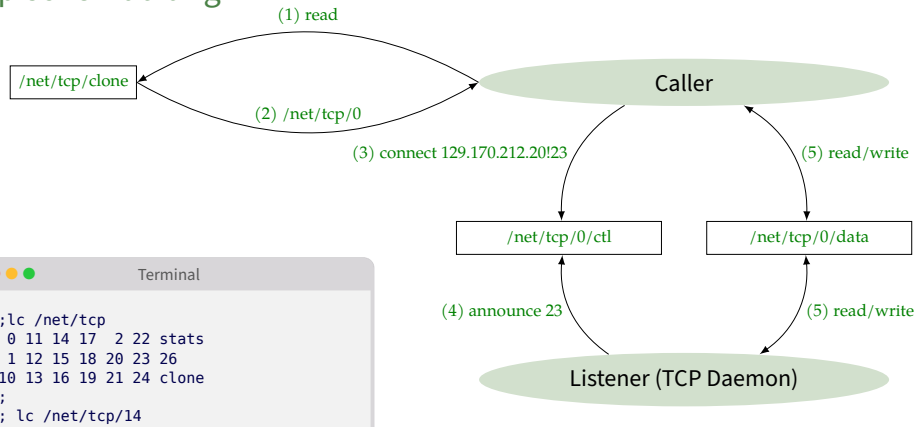
- Bildschirm-Kopie drucken

Terminal
```
; lp /dev/screen
```

- DNS-Abfrage

Terminal
```
; echo osg.informatik.tu-chemnitz.de!http > /net/dns
```


Typische Nutzung



Terminal
```
;lc /net/tcp
 0 11 14 17  2 22 stats
 1 12 15 18 20 23 26
10 13 16 19 21 24 clone
;
; lc /net/tcp/14
ctl   data   err    listen
local remote status
```


9P Protokoll

- Das eigentliche (nutzertransparente) Kernstück von Plan 9
  - Häufige Aussage: „Plan 9 ist ein File-Protokoll mit einer Handvoll Servern, die dieses Protokoll implementieren.“
- 9P ist ähnlich NFS
- Leichtgewichtig:
  - 17 Nachrichtentypen ➡ 3 für Authentifizierung, 14 für Objektmanipulation
  - Beschreibung 14 Seiten (Vergleich: NFS 4.0: 275 Seiten)
- Neben 9P gibt es spezialisierte Protokolle: exportfs, rexexec, ncpu, venti

Privater Namensraum

- Einen globalen Namensraum zu erhalten ist schwer in einem verteilten Mehrnutzersystem
  ➡ Verzicht, statt dessen private Namensräume
- Namen sind **privat** für jede Prozessgruppe, z.B.:
  - Prozesse eines Nutzer einer Maschine
  - Prozesse, die in einem Fenster eines Fenstersystems laufen
- Conceptual Model: private UNIX-Mount-Tabellen für jeden Prozess
  - Vorteil: `/dev/mouse` meint immer „meine“ Mouse
- Jedes Nutzerprogramm kann Namensraum anpassen:
  - `bind`: Einblenden anderer Teile der existierenden Namenshierachie (statt linken)
  - `mount`: Einbinden von Servern via 9P
  - `import`: Import fremder Namensräume

Beispiel: Netzwerk-Zugriff

- Zugriff auf ein Netzwerkinterface im internen Bell Labs Netwerk:

Terminal

```
% telnet dartmouth.edu
telnet: Cannot translate address dartmouth.edu
% import internetgate.bell-labs.com /net
% telnet dartmouth.edu
connected to tcp!dartmouth.edu!telnet on /net/tcp/15
UNIX(r) System V Release 4.0 (norob) (pts/1)
login:
```


Ungewöhnliche Fileserver

- Dienste sind stets **Fileserver**
  - Kernel- oder Nutzerraumdienste sind für den Klienten nicht unterscheidbar
  - Window System rio (Vorgänger: $8^1/_2$) ist als Fileserver implementiert
    - Nutzer sieht Fenster mit laufender Anwendung
    - Klientenanwendung liest File in /dev wie z.B. /dev/mouse, /dev/keyboard etc. und schreibt nach /dev/cons
  - Zentraler Sicherheitsserver (Standard: Factotum)
  - etc.
- Konsistenz wird vom Fileserver gelöst, nicht vom Protokoll
  - **Aber:** IL (internes leichtgewichtiges Transportprotokoll) ist reihenfolgetreu ➡ synchron

Resümee

Erkenntnis

- Klare Abstraktion (alles ist ein File) hilft, Verteiltheit zu bewältigen

- **Außerdem interessant bei Plan 9:**
  - Behandlung von Heterogenität: Einheitliches Namensschema für Programme und Unicode für Daten
  - Kein root-Nutzer, sondern File-Rechte für unterschiedliche Administrationsaufgaben
  - Zeitstempel als Pfadnamen
  - Programmiersprachen **Alef** und **Limbo** mit Unterstützung von Parallelität/Verteiltheit
  - `proc`-Filesystem stammt aus Plan 9

Spin-offs und verwandte Projekte

- **Inferno**
  - kleines Plan 9, u.a. für eingebettete Systeme
  - läuft nativ, als Subsystem in anderen OS sowie in Web-Browsern
- **9grid**
  - Global verteilte Installation von Plan 9 Sites für „Grid-Aktivitäten“
- **Plan B**
  - Plan 9 Derivat mit Mechanismen zur automatischen Adaption an Umgebungsänderungen
  - Einfluss vom Off++ Kernel

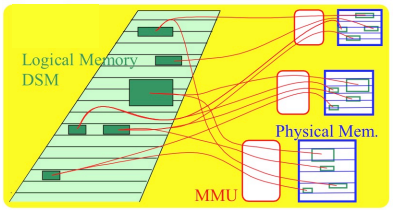
3.4 Plurix – wir machen eine Transaktion

- Cluster-Betriebssystem[1]
- Entwickelt an der Universität Ulm seit 1994
  - Ursprünglich verteiltes Oberon-System
  - Nachfolger **Rainbow OS**, 64 Bit
- Java-basiert
  - vollständig in Java geschrieben, auch Kernel und Gerätetreiber
  - Betriebsmittel sind Java-Objekte
  - eigener Java-Compiler
- **Gemeinsamer virtueller** Speicherraum über alle Maschinen eines Clusters
- Klein:
  - $<$ 20.000 LoC
  - Startup-Zeit einer einzelnen Station: $<$ 1s

[1] nicht zu verwechseln mit dem gleichnamigen UNIX-clone der University Rio de Janeiro

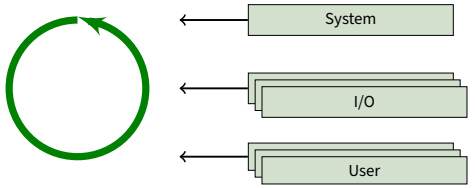
Gemeinsamer Speicher

- Distributed shared memory (*DSM*)
- Besitzer einer Seite verwaltet Original, ggf. Versand an andere Knoten
- **Lesezugriff**
  - Anfordern der Seite vom Eigner (falls nicht lokal vorhanden)
- **Schreibzugriff**
  - Anfordern der Seite vom Eigner (falls nicht lokal vorhanden)
  - Erzeugung einer Schattenkopie der Seite (Original)
  - Schreiben auf Originalseite
- **Konfliktauflösung** über ➡ **Transaktionsmodell**



Transaktionsmodell

- Alle **Aktionen** laufen als Teil einer Transaktion
- Optimistische Implementierung ➡ kurze Transaktionen / geringe Konfliktwahrscheinlichkeit



- Serialisierung der Transaktions-Auswertungsphase
- Vergleich des Write-Sets der auszuwertenden Transaktion mit den Read-Sets aller noch laufenden Transaktionen
- Abbruch der noch laufenden Transaktionen im Konfliktfall

Resümee

Erkenntnis

- Verteilter gemeinsamer Speicher ermöglicht Ortstransparenz
- Verteilter Speicher ist eine „angenehme“ Abstraktion
- Transaktionsmodell schlichtet Konflikte

- **Außerdem interessant bei Plurix:**
  - Keine IPC (außer Speicher)
  - Smart Buffers
  - Back-Chaining unterstützt Garbage Collection
  - Persistenz-Server über Namensdienst
  - Fehlertoleranz durch Klasserredundanz und Persistenz (Pageserver)

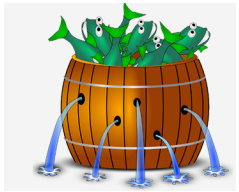
Andere DSM-Systeme

- **IVY**
  - Princeton University, 1986
  - Seitenbasiert
- **Mirage**
  - University of California at LA, 1988
  - Seitenbasiert
- **Munin**
  - Rice University, 1989
  - Objektbasiert

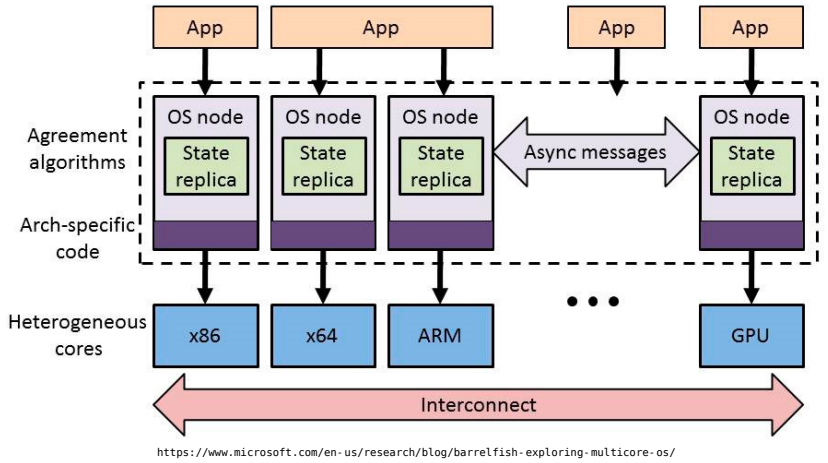
3.5 Barrelfish

- Aktives Forschungsprojekt von ETH Zürich + Microsoft Research, später auch HP Huawei, Cisco, Oracle, und VMware
- Ausgangspunkt sind neue Hardware-Herausforderungen: Zunehmende Heterogenität, alles ist NUMA, überall Latenzen, aufwändige Cache-Kohärenz, kaum Kontinuität

- Grundprinzipien von Barrelfish
  - Kommunikation ist explizit
  - Zustand wird repliziert oder partitioniert, aber nicht geteilt.
    - Gilt insbesondere bei *Multicore*-Prozessoren
    - Realer geteilter RAM/Cache nur als Optimierung
    - Invarianten durch verteilte Algorithmen ➡ keine Sperren
  - Abstraktionen funktionieren unabhängig von der Hardware



Architektur



https://www.microsoft.com/en-us/research/blog/barrelfish-exploring-multicore-os/

Architektur (Forts.)

- Minimaler Betriebssystemkern (**cpu driver**) pro CPU-Kern
  - Spezifisch für die jeweilige Architektur (X86, ARM, GPU, ...)
  - Verwaltung von **Fähigkeiten**
  - Ausführung von Systemrufen
  - Interruptbehandlung und Ausnahmebehandlung
  - Scheduling von **Dispatcher**-Instanzen
    - Entspricht einem klassischen Prozess
    - Auf mehreren Knoten, bilden die verteilte Anwendung (**domain**)
    - Scheduler ruft *Dispatcher* bei Aktivierung auf (**upcall**)
    - Keine Migration
- **Fähigkeiten** (*capabilities*)
  - Hierarchische Zugriffskontrolle für Ressourcen
  - Werden nur vom Kern modifiziert, Anwendung erhält Referenzen
  - Typisiert: RAM, Geräterahmen, Seitentabelle, Seitenrahmen, Endpunkt, ...

Dienste und Kommunikation

- *Dispatcher* einer Anwendung nutzen gemeinsam Ressourcen
- Notwendiger Nachrichtenaustausch über **Kanal**-Konzept
- Asynchrones Kommunikationsmodell mit entfernten Prozedurruf
- Dedizierte Sprache zur Schnittstellenbeschreibung
- Betriebssystem als Sammlung von kommunizierenden Diensten
  - Debugging, Gerätetreiber, Sperren, Speicherverwaltung, Netzwerk, Stromsparfunktionen, Terminal , ...
  - Werden jeweils als *Dispatcher* realisiert
  - Diverse Abhängigkeiten, Gerätetreiber verwenden bspw. Dienst für Verwaltung von Fähigkeiten
- *System Knowledge Base* Dienst - Prolog-basierte Verwaltung von Treibern, Interrupts, PCI-Geräten und Sperren

3.6 Zusammenfassung

- Verteilte Betriebssysteme für transparenten Zugriff auf verteilte Ressourcen
- Unterschiedliche Abstraktionen zur Erreichung unterschiedliche Ziele
- Bisher hat sich keine Standardparadigmen durchgesetzt, aber viele Ideen flossen in den Mainstream
- Klassisches Thema in der Forschung zu verteilten Systemen

Literatur

[Sin97] Sinha , P. K. : *Distributed Operating Systems*. IEEE Press, 1997

[Loe92] Loepere , K. : *Mach 3 Kernel Principles*. Open Software Foundation, 1992

[BIK99] Bischof , H.-P. , G. Imeyer und B. Kühl: *Das Netzbetriebssystem Plan 9: Konzepte und Programmierung*. Hanser, 1999

[Goe+04] Goeckelmann , R. , M. Schoettner, S. Frenz und P. Schulthess: „Plurix, a distributed operating system extending the single system image concept“, in *Electrical and Computer Engineering, 2004. Canadian Conference on*, 1985 - 1988 Vol.4

[Bau+09] Baumann , A. , S. Peter, A. Schüpbach, A. Singhania, T. Roscoe, P. Barham und R. Isaacs: „Your computer is already a distributed system. Why isn't your OS?“, in *Proceedings of the 12th Workshop on Hot Topics in Operating Systems*, Monte Verità, Switzerland