E MACH

E.1 Überblick

- Entwicklung seit 1986 an der Carnegie Mellon Univ. (CMU)
- später Basis für OSF

1 Motivation

- UNIX-Systemkern heute nicht mehr so einfach aufgebaut und leicht modifizierbar wie früher
  - früher auf PDP11 mit 32k HSP lauffähig, heute 1,5 - 2 MB Speicher (inkl. Puffer etc.) notwendig
  - viele neue Funktionen wurden kritiklos in den Kern eingebaut, weil dort die notwendigen Information leicht zugänglich sind

➥ Aufblähung des Kerns, Abstraktionen und Strukturierung werden mehr und mehr zerstört

1 Motivation (2)

- für Einsatz von UNIX auf Multiprozessorsystemen sind aufwendige Modifikationen notwendig
  - Kern-Monitor muß in Teile zerlegt werden oder ganz aufgegeben werden
  - an Stellen, an denen die Monitor-Eigenschaft aufgegeben wird, ist aufwendige Koordinierung erforderlich
- Benutzer- oder anwendungsspezifische Betriebssystemmechanismen werden immer häufiger gefordert
  - spezielle Pufferstrategien für Datenbanken
  - spezielle Blockgrößen in Dateisystemen für Bildverarbeitung
  - spezielle Zugriffsschutzmechanismen im Dateisystem
  - …

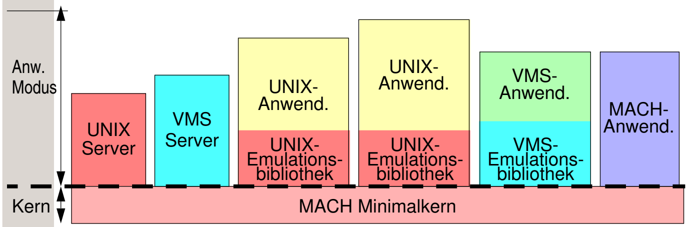
2 wesentliche Eigenschaften

- Unterstützung für Multiprozessoren
- neues - hardwareunabhängiges - Konzept der virtuellen Speicherverwaltung
- *Capability*-basierte Interprozeßkommunikation - transparent erweiterbar für Netzwerk-Kommunikation
- Modularer Aufbau, einfache Erweiterbarkeit, nur die wichtigsten Funktionen sind im MACH-Kern realisiert

  ➥ **policy - mechanism separation**
- die Betriebssystemumgebung wird durch ***Server*** realisiert, die über die Basis-Mechanismen des MACH-Kerns angesprochen werden können
  - ➤ Dateisystem
  - ➤ Netzwerk-Kommunikation (z. B. TCP/IP)
  - ➤ Scheduling

E.2 Architektur

- Minimalkern als Emulationsbasis für UNIX und andere Betriebssysteme
- eine Betriebssystemulation besteht aus zwei Teilen:
  - eine Komponente im virt. Adreßraum der Anwendung
  - einer Menge von Server-Prozessen
- Verschiedene Betriebssysteme können gleichzeitig auf MACH residieren
  - z. B. BSD UNIX, SystemV UNIX, HP-UX, VMS



E.3 Basis-Abstraktionen des MACH-Kerns

1 *Task*

- **Umgebung für Programmausführungen**
  - ◆ virtueller Adreßraum
  - ◆ Verwaltungseinheit für den Zugriff zu Systemressourcen
    - ➤ Prozessoren
    - ➤ Port-Capabilities (= Zugriffsrechte)
    - ➤ Speicherbereiche
  - ◆ passiv

2 *Thread*

- **ein Aktivitätsträger innerhalb einer *Task***
  - ◆ besteht aus
    - ➤ einem unabhängigen Programmzähler
    - ➤ einem Registersatz
    - ➤ eigenem Stack-Bereich im virtuellen Adreßraums der *Task*
- alle *Threads* einer *Task* haben gemeinsamen Zugriff zu allen *Task*-Ressourcen

➥ **Aufspaltung des traditionellen Prozeß-Paradigmas in zwei orthogonale Konzepte**
- **Aktivitätsträger**
- **Ausführungsumgebung**

3 *Memory Object*

- **Basiselement der Speicherverwaltung**
  - ◆ Segment in einem virtuellen Adreßraum
  - ◆ kann gleichzeitig in mehrere Adreßräume abgebildet sein (= shared memory)
  - ◆ Ein-/Auslagerung der Seiten erfolgt über den ***Pager*** des Memory Objects (kann auch eine normale Anwendung sein)

4 *Port*

- **Kommunikationskanal - Warteschlange für Mitteilungen**
  - ◆ wird vom MACH-Kern verwaltet
  - ◆ geschütztes Objekt
  - ◆ Port-Referenzen haben **Capability**-Eigenschaften

5 *Message*

- **Menge von Daten-Objekten zur Kommunikation zwischen *Threads***
  - ◆ werden über Ports verschickt
  - ◆ können max. den gesamten virt. Adreßraum umfassen
  - ◆ besitzen einen Typ, können Zeiger auf Daten und *Capabilities* für *Ports* enthalten (wird vom Kern interpretiert!)

E.4 Systemschnittstelle

- Abstraktionen des MACH-Kerns (Tasks, Threads, …) und externer Betriebssystem-Server (Dateien, TCP-Ports, …) werden durch Softwareobjekte (Tasks, Threads, etc.) repräsentiert
- Zur Kommunikation mit diesen Objekten werden Ports eingerichtet
- Operationen werden durch Messages an diese Ports ausgeführt

Beispiel: Beim Generieren einer *Task* erhält man die Zugriffsrechte auf einen *Port* dieser *Task*.
Durch *Messages* an diesen *Port* kann die *Task* beeinflußt werden.

- Jede Task hat initial Zugriff auf einen **Task-Port** (Port des Kerns!) für Systemaufrufe und einen **Bootstrap-Port** für andere Interaktionen
  - ◆ UNIX-Anwendungen benutzen den Bootstrap-Port zur Kommunikation mit dem UNIX-Server

E.4 Systemschnittstelle (2)

- Ausnahmesituationen (Adressierungsfehler, Division durch 0, …) werden vom Kern durch Messages an einen speziellen **Exception-Port** der auslösenden Task mitgeteilt
- Server-Anwendungen erhalten über zwei spezielle, priviligierte Ports direkten Zugriff auf die Systemressourcen:
  - ◆ **Device-Port** generische Geräteschnittstelle
  - ◆ **Host-Port** priv. Systemschnittstelle (Systemzeit setzen, Prozessormanipulationen, booten, etc.)