

Universität Osnabrück  
Informatik  
Informatik  
SS 2000  
Microkernel  
Leitung: A. Polze

# Der GNU HURD & Mach-Architektur

Britta Koch  
Am Salzmarkt 3, WG 11  
49074 Osnabrück  
M.A. Computerlinguistik und Künstliche Intelligenz  
CL/KI(6) / Philosophie(6) / Informatik(6)

# 1 Informationen über HURD

- Akronym: HURD = Hird of UNIX-replacing daemons, HIRD = Hurd of interfaces representing depth - gegenseitig rekursives Akronym
- von Debian “Distro” rund um den Hurd - kein Uni-Spielzeug, soll Linux-kompatibel sein
- basiert auf GNU Mach, mit eigenen Servern (= Translators)

## 1.1 Geschichte

- 1983 von RMS gestartet (GNU) - um komplett freies OS zu haben - primitiver Kernel
- 1986: Verhandlungen, einen Mach-Kernel zu nehmen
- 1990: Linux, aber 1991 HURD offizieller GNU-Kernel

## 1.2 Stand der Dinge

- runterladen, mit GRUB-Bootloader installieren
- noch nicht sehr usable...
- hat gcc, dpkg, viele .debs, X!
- hat noch nicht: ppp, Sound, IRQ-Sharing, mehrere Netzwerkkarten, ...

## 1.3 Vergleich mit Linux

- Vorteile: OO-Struktur, skalierbar (SMP), erweiterbar (Struktur), stabil (kein Reboot bei neuem Kernel)
- Linux: wenig dokumentiert, Low-level Memory Management, higher-level Networking - alles im Kernel - monolithisch, bei z. B. neuem Netzwerk-Protokoll im Kernel selber implementieren, kein Framework
- Mikrokernel-Vorteile: nur das nötigste im Kernel selber (Prozesse, Memory Management, Scheduling, Interrupts, basic I/O), Rest im User-Space, klare Interfaces - selbst Kernel austauschbar (anderes Memory Management, Scheduling)
- nicht nur für PCs gedacht
- vielleicht weniger effizient, dafür elegantere Struktur (Assembler!)

## 1.4 Geschichte von Mach

- 1975 RIG (Rochester Intelligent Gateway) - Message Passing, Modulare OS-Struktur
- 1975: einer ging zur Carnegie-Mellon Uni, auf modernerer Hardware: Accent
- 1984: 3. Generation OS - Mach, Unix-kompatibel
- mit DARPA und BSD-Kernel
- 1988: Mach wird Mikrokernel - BSD-Code raus

## 2 Das HURD-System

### 2.1 Installation

- GRUB runterladen, .tar.gz runterladen, entpacken
- extra Partition ( $\leq 1$  GB) mit ext2fs
- native-install skript - .debs entpacken
- devices anlegen

### 2.2 Wie sieht es aus?

- fast wie Linux - ohne extra .debs amerikanische Tastatur-Belegung, kein gcc, X
- /usr symbolischer Link auf /
- für Netzwerkkarte kein ifconfig, sondern Translator

### 2.3 Wie funktioniert es?

- Booten: mit GRUB Mach-Kernel angeben
- serverboot lädt den Kernel (Multiboot-enabled, Linux, BSDs nicht)
- mit serverboot auch andere Kernel starten, zum testen - keine Devices teilen!
- sehr multithreaded
- Ports zum Kommunizieren im Kernel (Unix fd / Sockets)
- Translators für I/O

## 3 Mach- und HURD-Architektur

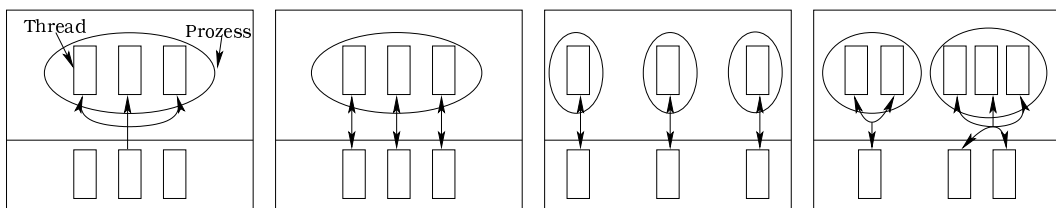
- Mach: Unix-Emulation im User-Space - portierbar
- OO: zwar in C, und keine Vererbung, kennt aber die Klassen Port ( $\rightarrow$  Sockets), Nachricht, Task (= Prozess), Threads, und Virtual Memory
- Authentifizierungsserver (Sicherheit bei Kommunikation mit anderen Tasks), Prozeßserver (CORBA: Nameserver) - andere nach Wunsch dazu (NFS, Sockets, ...)
- Translators: Mountpunkte, symbolische Links, Devices
- jede Datei hat einen Translator
- FTP-Translator, Datenbank, ...

### 3.1 Tasks

- Umgebung, in der eine Ausführung stattfinden kann: Adreßraum (Programtext, Daten, Stacks) - Basiseinheit für Zuteilung von Betriebsmitteln
- passiv - Ausführung an Threads gebunden, Container für Threads
- jede Task hat bestimmte Ports: Prozeß-Port (Kommunikation mit Kernel - Systemaufrufe), Bootstrap-Port (Initialisierung), Exception (Fehlermeldungen - Debugger!), registrierte Ports (Komm. mit Standard-System-Servern)
- Prozesse können rechenbereit oder wartend sein - unabhängig von Threads!
- Scheduling-Parameter (SMP)
- besitzt, anders als Unix-Prozesse, keine uid, gid, Wurzel- und Arbeitsverzeichnis, kein Feld von Dateideskriptoren

### 3.2 Threads

- ausführbare Einheit (Programmzähler, Register) - gehört zu genau einem Prozeß
- alle Prozesse eines Threads benutzen den Adreßraum des Threads gemeinsam
- jeder Thread eigenen Thread-Port (Systemaufrufe, wie Beenden)
- können auf Ports der Geschwister zugreifen - Kontrolle!
- bei SMP mehrere gleichzeitig aktiv - Synchronisation, Scheduling
- lauffähig / blockiert
- C-Threads Paker: API zu Kernel (fork, exit, join, yield, detach)



Alle C Threads benutzen einen Kernel Thread.

Jeder C Thread hat seinen eigenen Kernel Thread.

Jeder C Thread hat seinen eigenen Single-Thread-Prozess.

Beliebige Zuordnung von User Threads auf Kernel Threads.

- Zuordnung Threads / Prozesse:
  - alle C-Threads in einem Kernel-Thread - ein Thread blockiert ganzen Prozess
  - ein Mach-Thread pro Kernel-Thread (auch m zu n)
  - ein Thread pro Prozess - Ressourcen (Ports, File Handles) teilen unmöglich
  - beliebig viele User-Threads auf beliebig viele Kernel-Threads - am flexibelsten, aktuell

### 3.3 Memory

- Speicherobjekt: Datenstruktur, im Adreßraum eines Prozesses abgebildet
- Basis für Verwaltung des virtuellen Speichers
- maschinenunabhängigen Teile der Speicherverwaltung klar getrennt von maschinenabhängigen - Portabilität!
- Speicherverwalter (externer Pager) = Benutzerprozeß - logische Verwaltung
- Benutzer können eigene schreiben
- Prozesse können Speicherobjekte gemeinsam benutzen (SMP!)
- bei Erzeugen von Kindsprozessen: zunächst als read-only geteilt, bei Schreiben des Kindes kopiert (copy-on-write)
- externer Pager: zum Mappen 3 Ports - Objekt-Port (Pager - Info), Kontroll-Port (Kernel - Ereignisantwort), Namens-Port (Kernel - Nachfragen anderer)
- Paging-Dämon - überprüft regelmäßig den Zustand des Speichers und veranlaßt externe Pager zum Swappen

### 3.4 Ports

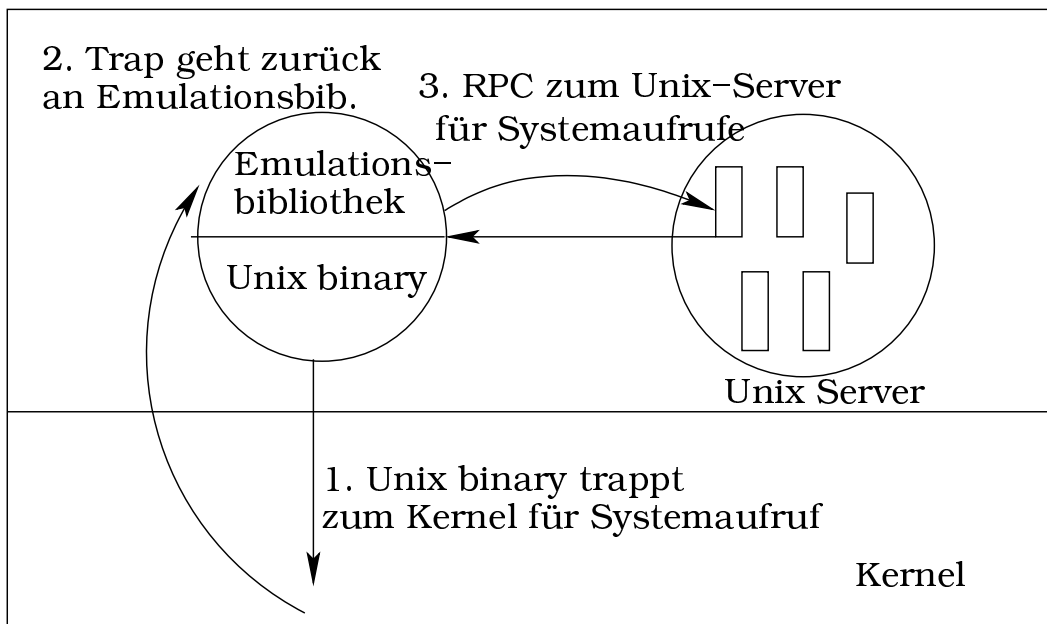
- geschützter Briefkasten für Prozesse
- speichert geordnete Liste von Nachrichten
- wie Pipes in Unix - unidirektionale Kommunikation
- Nachrichten gehen garantiert nicht verloren und kommen in der richtigen Reihenfolge an
- nicht als Byte-Ströme - strenge Grenzen zwischen Nachrichten
- Nachrichten extra als Nachrichtenschlange gespeichert
- in Portgruppen zusammenfassen - nur lesend

### 3.5 Nachrichten

- einfache oder komplexe Nachrichten
- komplexe Nachrichten: Folge von Deskriptor-Datenfeld-Paaren
- out-of-line-Daten: Daten ohne zu kopieren übertragen
- Ports in komplexe Nachrichten verpacken

### 3.6 Unix-Emulation

- Unix Emulator als Server auf Mach
- Unix Server, Systemfunktionsemulationsbibliothek
- Unix Server an Kernel: trap Systemaufruf und schicke an Emulationsbibliothek (Trampolin)
- Emulationsbibliothek: RPC an Unix Server zum Arbeiten



- Unix Server mit C-Threads implementiert
- I/O Aufrufe anders: Performance
- Datei öffnen ohne RPC - direkt von Emulationsbibliothek
- eigenen Memory Manager (Inode-Pager)
- kopiert Dateien in Applikationsspeicher um - Synchronisation bei gleichzeitigem Zugriff auf eine Datei
- klingt alles langsam, ist es aber nicht
- demnaechst Unix-Server aufsplitten?

### 3.7 Translators

- Dateisystem: hierarchisch geordnete Dateien und Ordner
- mit Pfad beschrieben
- außerdem harte und weiche Links, Devices, FIFOs
- viele gleiche Eigenschaften: Besitzer, Gruppe, Zugriffsrechte - alles in Inodes
- alle Inodes mit read() und write() lesen
- im HURD anders: Translator zwischen Datei / Inode und Benutzer, der darauf zugreift
- Angaben über Translators in der Inode gespeichert - mit User-Rechten darauf zugreifen (Translator-UID = Owner des Files)
- Mount-Points = Inode mit besonderem Translator
- Devices, Symbolic Links

- aktive und passive Translators
- passive werden erst bei Zugriff gestartet (→ modprobe)
- Programme settrans, showtrans
- mount sollte theoretisch ein Shell-Skript sein (ist aber nicht)
- damit Komplexität von ftp, tar, gzip, XDisplays verstecken...
- Servers sind Translatoren
- werden in .def-Dateien definiert
- GNU-C-Library verwendet für Unix System Calls
- Netzwerk auch mit Translator: pfinet (passiv)
- jedes Programm als Translator!
- Authentication-Server: RPC, Schnittmenge von 2 Authentication Ports
- auch wenn User Authentication-Server schreiben - Programme reden mit denen, denen sie vertrauen - wenn nicht der gleiche, nix Auth.
- deshalb bloss ein Auth-Server - aber extra Passwort-Server
- Process-Server: uname -a, POSIX, Mach tasks ↔ HURD Prozesse, PIDs, Message Ports registrieren
- verschiedene Prozess-Server...
- gibt auch crash-server (bei segfault): suspend / kill / core dump
- exec-server: aus image file den Prozess herstellen (boot-floppies)
- fifo-server, firmlink (zw. symlink und hard link), ftpfs, fwd (zum forwarden an andere Server: fifos), init, isofs, magiv, nfs, null, pflocal (für pipes), symlink, term, ...