Konzepte von Betriebssystem-Komponenten

WS 2002/2003

Microkernel-Betriebssysteme (Mach, L4, Hurd)

Holger Ruckdeschel
holger.ruckdeschel@informatik.stud.uni-erlangen.de

14. November 2002

1 Betriebssystem-Architekturen

Traditionelle Varianten von Unix und anderen Betriebssystemen sind nach dem Prinzip des *monolithischen Kernels* aufgebaut. Hierbei bilden alle Komponenten (Prozess- und Speicherverwaltung, Gerätetreiber, Dateisysteme, Netzwerkprotokolle, etc.) eine Einheit und laufen in einem gemeinsamen Speicherbereich ab. Dies hat den entscheidenden Nachteil dass keinerlei Schutz zwischen den Kernkomponenten gegeben ist. Schon kleinste Programmierfehler (falsche Zeiger usw.) können ein Fehlverhalten oder einen Absturz des Kernels und damit des gesamten Systems zur Folge haben. Insbesondere Treiber von Drittherstellern, die also nicht in Absprache mit den Kernel-Programmierern entwickelt werden, stellen so ein ernsthaftes Problem dar.

Natürlich sollen auch die Vorteile eines solchen Konzeptes nicht verschwiegen werden:

- Durch die hohe Integration der Kernkomponenten läuft die Kommunikation unter ihnen in der Regel extrem schnell und einfach ab.[1]
- Da alles im sogenannten *Kernel-Mode* läuft, ist es an jeder Stelle möglich direkt auf die Hardware zuzugreifen und privilegierte Maschinenbefehle auszuführen.[2]

Dieser letzte Punkt ist jedoch nicht unbedingt als Vorteil anzusehen, da er die oben genannten Stabilitäts- und Sicherheitsprobleme vor allem in Verbindung mit fremden Treibern zur Folge haben kann.

Um die Probleme des monolithischen Aufbaus zu lösen bietet sich folgende Vorgehensweise an: Sämtliche Komponenten die nicht zwingend besondere Privilegien benötigen werden aus dem Kernel entfernt und in gewöhnliche Benutzerprozesse („*Server*") ausgelagert. Das Ergebnis ist ein sogenannter *Microkernel*, der idealerweise nur noch eine sehr kleine Menge an Grundfunktionen zur Verfügung stellt:

- einfaches Prozessmanagement
- einfaches Speichermanagement
- *effiziente* Wege zur Inter-Prozess-Kommunikation (IPC)

Die restliche Funktionalität (Gerätetreiber, Dateisysteme, Netzwerkprotokolle, etc.) wird in Servern realisiert. Werden hierbei alle Komponenten in einen einzigen Server integriert, so spricht man vom *single-server*-, andernfalls vom *multi-server*-Betriebssystem. Zwischen den Servern findet ein geregelter Datenaustausch über IPC-Mechanismen statt.

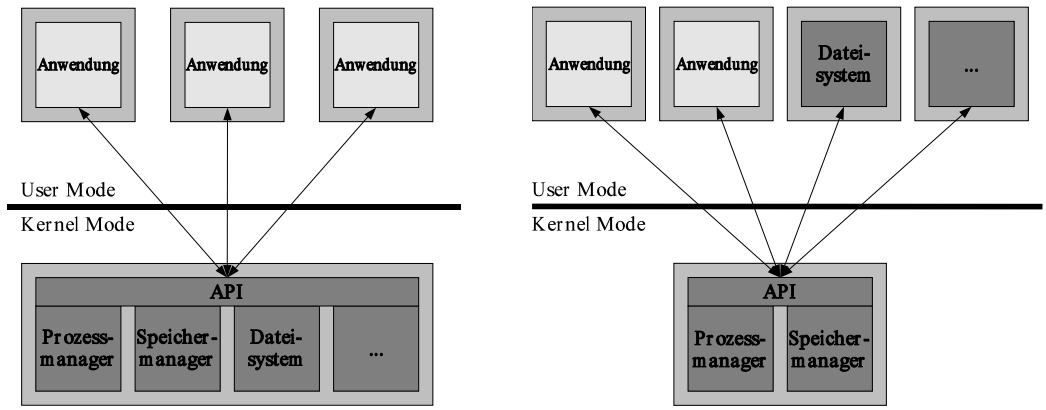
[1] z.B. über globale Variablen
[2] Im *User-Mode*, in dem sämtliche Anwendungen usw. laufen, ist dies nicht möglich.

Die Vorteile eines solchen Konzeptes liegen auf der Hand:

- Trennung der Systemkomponenten und damit Zugriffsschutz untereinander. Fehlerhafte Treiber können das System nicht mehr negativ beeinflussen.
- Fehlerbehebung ist im User-Mode wesentlich leichter als im Kernel-Mode.
- Leichte Erweiterbarkeit durch Modularisierung und standardisierte Schnittstellen.

Der wesentliche Nachteil einer solchen Architektur ist die geringere Geschwindigkeit. Durch die Trennung ist ein häufiges, zeitraubendes Umschalten[3] zwischen einzelnen Prozessen sowie dem Kernel notwendig. Das erhöhte Aufkommen an IPC verlangt nach einer besonders effizienten Implementierung solcher Mechanismen gegenüber monolithischen Systemen.

Die folgende Abbildung zeigt einen schematischen Vergleich der beiden Architekturen.



(a) Monolithischer Kernel

(b) Microkernel

Abbildung 1: Betriebssystem-Architekturen

In den nächsten Abschnitten sollen einige Beispiele zu Microkernel-Systemen vorgestellt werden.

2 Mach

2.1 Entwicklung

Der Mach Microkernel wurde an der Carnegie Mellon University (CMU) ausgehend von 4.2BSD Unix entwickelt. Ziel war es einen einfachen, flexiblen Kernel zu schaffen, der auf vielen verschiedenen Architekturen laufen sollte. So wurde Stück für Stück alter 4.2BSD-Code durch Mach-Code ersetzt. Auch die Neuerungen von 4.3BSD flossen ein. Mach Release 2 war ein vollständig BSD-kompatibler Kernel, der teilweise aus Mach, teilweise aus 4.3BSD-Komponenten zusammengesetzt war und entsprach noch dem traditionellen monolithischen Ansatz.

Erst mit Release 3 kam die Umstellung auf Microkernel. Sämtliche BSD-Funktionalität wurde in externe Server ausgelagert. Auf Basis dieses Unix-freien Mach 3 Kernels können nun auch andere Betriebssysteme wie z.B. DOS, MacOS oder OSF/1 laufen. Solche Systeme können sogar gleichzeitig auf einem Rechner betrieben werden, womit das Konzept der virtuellen Maschinen in Software umgesetzt wurde.

[3] sog. *Kontextwechsel*

2.2 Aufbau

Als Microkernel stellt Mach nur eine geringe Menge einfacher Abstraktionen bereit:

- Ein *Task* ist eine Umgebung zur Ausführung von Prozessen. Er besteht aus einem virtuellen Adressraum, einem oder mehreren Ports zur Kommunikation sowie mindestens einem Thread.
- Als *Thread* wird ein einzelner Ablauf innerhalb eines Tasks bezeichnet. Selbstverständlich kann ein Task mehrere nebenläufige Threads enthalten. Diese verwenden dann alle zum Task gehörenden Resourcen (Speicher, Ports, etc.) gemeinsam.
- Ein *Port* ist ein einfacher Kommunikationskanal. Der Zugriff auf Ports wird vom Kernel über Rechte (*port rights*) kontrolliert. Mehrere Ports können zu einem *Port Set* zusammengefasst werden.
- Über Ports werden *Nachrichten* (*messages*) zwischen Tasks und dem Kernel ausgetauscht. Dies kann direkt oder auch indirekt über Zeiger geschehen.
- Über *Memory Objects* wird den Tasks Speicher zur Verfügung gestellt. Neben physikalischem Speicher kommen auch Dateien oder Pipes als Memory Objects in Frage. Diese werden von externen Servern (z.B. Dateisystem-Server) bereitgestellt und in den Adressraum des Tasks eingeblendet.

Der Mach Microkernel ist zwar sauber aufgebaut, enthält jedoch eine Reihe an Abstraktionen (z.B. I/O-System) die besser dezentral in Servern aufgehoben wären. Daher soll im folgenden Abschnitt ein weiterer Microkernel vorgestellt werden.

3 L4

3.1 Entwicklung

Im Gegensatz zu Mach wurde der L4 Microkernel von Anfang an mit Blick auf Minimalität entwickelt. Die ursprüngliche Version L4/x86 stammt von Jochen Liedke und läuft ausschließlich auf x86-Prozessoren. Mittlerweile existieren jedoch L4-Implementierungen für verschiedenste Architekturen, z.B. Alpha, MIPS, StrongARM, PowerPC sowie IA64. Da alle einer gemeinsamen Spezifikation gehorchen soll im Folgenden nur auf den architekturunabhängigen Aufbau von L4 eingegangen werden.

3.2 Aufbau

L4 kennt die folgenden primitiven Objekte:

- *Flexpages* sind flexibel große Speicherseiten. Über sie kann auf den Hauptspeicher und auf den E/A-Speicher von Peripheriegeräten zugegriffen werden.
- *Virtuelle Adressräume* sind aus Flexpages zusammengesetzt. Für das Ein- und Ausblenden von Flexpages stehen die folgenden Operationen zur Verfügung:
  - `grant`: Die Speicherseite wird an einen anderen Task übergeben und steht dem ursprünglichen Besitzer nicht mehr zur Verfügung.
  - `map`: Die Speicherseite wird an einen anderen Task übergeben und steht anschließend beiden beteiligten Tasks zur Verfügung (shared memory)
  - `flush`: Die Speicherseite wird aus dem Adressraum aller Prozesse entfernt die diese vorher über `map` erhalten haben.
- *Threads* sind Abläufe innerhalb eines Adressraums.
- *Tasks* bestehen aus einem Adressraum in dem mindestens ein Thread abläuft.
- *Clans* sind Gruppen von Tasks. An der Spitze eines Clans steht der *Chief*-Task, über welchen alle IPC-Nachrichten laufen die die Clan-Grenzen überschreiten. Auf diese Weise lassen sich differenzierte Sicherheits-Konzepte realisieren.

3.3 I/O-Implementierung

L4 bietet selbst keinerlei Schnittstellen zur Ansteuerung von Peripheriegeräten an, sondern überträgt die Verantwortung komplett auf externe Server. Um dies zu realisieren kann auf die oben genannten L4-Funktionen zurückgegriffen werden:

- Die I/O-Adressräume der Hardware (I/O-Speicher und I/O-Ports) können in die virtuellen Adressräume der im User-Mode laufenden Gerätetreiber eingeblendet werden (`map` bzw. `grant`).
- Hardware-Interrupts werden vom Kernel aufgenommen und via IPC an den betreffenden Gerätetreiber weitergeleitet.
- Gleiches gilt für Seitenfehler (page faults) die beim Zugriff auf nicht im Speicher befindliche Seiten ausgelöst werden. Auf diese Weise lassen sich sogar Speichermanager im User-Mode implementieren.

4 Geschwindigkeitsvergleich

So vorteilhaft das Microkernel-Design auch sein mag, der wesentliche Nachteil solcher Systeme ist die geringere Geschwindigkeit. [5] verglich dazu folgende Systeme auf einem Pentium-133 PC:

- den normalen monolithischen Linux-Kernel
- MkLinux, ein auf Mach 3.0 aufsetzendes Single-Server Linux
- L^{4}Linux, welches auf L4 Version 2 basiert (ebenfalls Single-Server)

Als low-level Benchmark wurde die Zeit gemessen, die der einfachste System-Call, `getpid()`, benötigt.

Linux	1,68μs
MkLinux	110,60μs
L^{4}Linux	3,95μs

Tabelle 1: `getpid()`

Es zeigen sich also gravierende Unterschiede zwischen dem monolitischen Linux und der Mach-basierten Version. L^{4}Linux ist zwar deutlich schneller als MkLinux, benötigt aber immernoch mehr Zeit als reines Linux.

Da solche einfachen Messungen in der Regel wenig aussagekräftig sind, wurde als high-level Benchmark der Linux-Server unter allen drei Systemen kompiliert.

Linux	476s
MkLinux	605s
L^{4}Linux	506s

Tabelle 2: Kompilieren des Linux-Servers

Wiederum ist das monolitische Linux am schnellsten. Auch L^{4}Linux ist nur wenig langsamer, wohingegen MkLinux deutlich abfällt.

Beide Messungen lassen folgenden Schluss zu: Ein monolithischer Kernel ist schneller als ein Microkernel, jedoch spielt das Design des Microkernels eine entscheidende Rolle, denn L4 hat einen deutlichen Vorsprung gegenüber Mach.

5 GNU Hurd

Nun soll noch ein komplettes Microkernel-basiertes Unix-kompatibles Betriebssystem vorgestellt werden. GNU Hurd[4] ist eine Multi-Server Unix-Implementierung auf Basis eines abgewandelten Mach Microkernels (GNU Mach). Zusammen mit einigen Bibliotheken, darunter einer angepassten libc, einigen speziellen Utilities und einer großen Menge weiterer GNU-Software wird ein komplettes, freies Unix-Betriebssystem zur Verfügung gestellt.

5.1 Server in Hurd

Server, in Hurd auch als Übersetzer (translators) bezeichnet, werden mit dem `settrans` Kommando gestartet. Um beispielsweise eine ext2-Festplattenpartition zu mounten genügt folgender Aufruf:

```
# settrans -c /mountpoint /hurd/ext2fs /dev/hd0s1
```

Dieser Befehl befestigt am Verzeichnis `/mountpoint` den Übersetzer für das ext2-Dateisystem welchem die zu verarbeitende Festplattenpartition übergeben wird. Der Übersetzer läuft anschließend mit den Rechten des aufrufenden Benutzers und erscheint als normaler Prozess in der Prozesstabelle. Im Gegensatz dazu stellt das Linux-Äquivalent

```
# mount /dev/hda1 /mountpoint -t ext2
```

einen Befehl direkt an den Kernel dar die Partition zu mounten. Dies kann nur gelingen wenn das betreffende Dateisystem auch vom Kernel zur Verfügung gestellt wird. Ein Benutzer hat so keine Möglichkeit eigene Dateisysteme in das System zu integrieren. In Hurd kann jeder Benutzer seinen eigenen Übersetzer schreiben und starten und so das Betriebssystem unabhängig von anderen Benutzern gezielt auf seine Bedürfnisse zuschneiden.

Neben dem genannten `ext2fs`-Server bietet Hurd noch eine Vielzahl weiterer Server, z.B. `pfinet` (IP-Protokoll), `null` (null-Device), `mouse` (Maustreiber) u.v.a..

Literatur

[1] Silberschatz, Galvin, Gagne: Operating System Concepts (6th edition), John Wiley & Sons, Inc, 2001

[2] Hauck F., Hofmann F.: Systemprogrammierung I Folien, LS f. Informatik 4, Univ. Erlangen/Nürnberg, WS 2002/2003

[3] Liedtke Jochen: L4 Reference Manual, IBM T. J. Watson Research Center, 1996

[4] The L4 μ-Kernel Family, `http://os.inf.tu-dresden.de/L4/`

[5] Härtig, Hohmuth, Liedtke, Schönberg, Wolter: The Performance of μ-Kernel-Based Systems, Dresden University of Technology, Department of Computer Science, 1997

[6] The GNU Hurd, `http://www.gnu.org/software/hurd/hurd.html`

[7] Jährling Wolfgang: Hört Hurd, Linux Magazin 11/2002, Linux New Media AG, 2002

[4] Hurd: Hird of Unix-Replacing Daemons