Universität Karlsruhe
Fakultät für Informatik
Institut für Betriebs– und Dialogsysteme

MIKROKERN BETRIEBSSYSTEME IN PARALLELEN ARCHITEKTUREN

Diplomarbeit

Zusammenfassung einer Thesis in Computer Science

an der University of Massachusetts Dartmouth

von

Joachim Blum

Januar 1996

Zusammenfassung

MIKROKERN–BETRIEBSSYSTEME IN PARALLELEN ARCHITEKTUREN

von

JOACHIM BLUM

Diese Ausarbeitung faßt die wichtigsten Ergebnisse der Thesis Microkernel Operating Systems in Parallel Architectures *an der University of Massachusetts, Dartmouth, zusammen. Sie stellt die Notwendigkeit der Modularisierung von Betriebssystemen dar und erläutert, wie durch diese Modularisierung ein verteiltes Betriebssystem aufgebaut werden kann. An zwei Beispielen, Mach und Chorus, wird der Entwurf eines Mikrokern–Betriebssystems aufgezeigt. Schwachstellen dieser beiden Mikrokerne zur Nutzung als Basis eines verteilten Betriebssystems werden erläutert und konkrete Vorschläge zur Verbesserung v. a. in Hinsicht der Unterstützung für die Parallelverarbeitung gemacht.*

Danksagung

Ich danke hiermit für die Unterstützung von Professor Dr. Boleslaw Mikolajczak während der Erstellung der Thesis.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

In den vergangenen Jahren erhöhte sich die Betriebssystemkomplexität durch Hinzufügen von zusätzlichen Funktionalitäten. UNIX, als Beispiel, verlor die gute Strukturierung und wurde somit nicht portierbar für neue Architekturen. Die Betriebssystemkerne wurden so komplex, daß es sehr schwierig wurde, die Kerne zu unterhalten und zu verbessern.

Ein neuer Trend im Betriebssystementwurf ist, Dienste zu modularisieren und verschiedene Dienste, die bisher vom Kern angeboten wurden, in den Benutzerbereich zu extrahieren. Somit können wieder kleine, gut strukturierte Kerne entwickelt werden, die als Basis des Betriebssystems dienen. Diese Betriebssystemarchitektur wird als *Mikrokernarchitektur* bezeichnet.

Diese Mikrokerne beinhalten oft nur Interprozeßkommunikations– und Prozessorzuteilungsdienste. Der Großteil der bisherigen Betriebssystemdienste wird außerhalb des Kerns in eigenen Adreßbereichen realisiert. Diese Mikrokern–Betriebssysteme verringern nicht die Größe des Quellcodes, der zur Implementierung einer gewissen Funktionalität benötigt wird. Sie bieten nur einen strukturierten Rahmen für die Reduzierung der Komplexität des Entwurfs eines neuen Systems und der Einführung neuer Funktionalitäten in ein existierendes System. Diese Strukturierung bietet auch die Möglichkeit, die Dienste in einem parallelen System auf verschiedene Knoten zu verteilen, welches ein Grundbedürfnis des wachsenden Marktes von parallelen Systemen ist. Mikrokerne sind somit ein geeignetes Instrument der Softwaretechnik in der Betriebssystemerstellung. Mikrokerne bieten Systementwicklern Werkzeuge, die vorher nur Anwendungsentwicklern angeboten werden konnten.

Moderne Betriebssysteme müssen folgenden drei Softwareentwicklungsherausforderungen genügen, um wettbewerbsfähig zu sein:

- Unterstützung neuer paralleler Hardwarearchitekturen,
- Funktionalitätserweiterungen im Rahmen von offenen Systemen,
- Erweiterungen in Richtung einer kooperativen Plattform.

Systementwickler müssen neue Dienste in ein existierendes System integrieren können. Bei Mikrokernarchitekturen sind diese Erweiterung durch einfaches Hinzufügen von Dienstgebern möglich, die oberhalb des geschützten Mikrokerns existieren (Abbildung 1.1).

In bisherigen monolithischen Systemen existieren die Dienstgeber innerhalb des geschützten Bereichs. Somit wird durch jede Veränderung eines Dienstgebers ein neuer Kern benötigt. Als erschwerend wirken die starken Abhängigkeiten zwischen den einzelnen Teilen des Systems. Da für jede Änderung die Nebenwirkungen betrachtet werden müssen, sollte der Programmierer den Überblick über das komplette System haben, bevor er Änderungen vornimmt.

Mikrokern–Betriebssysteme bieten unterschiedliches Verhalten durch einfaches Austauschen von Dienstgebern. Die Dienstschnittstellen sind klar definiert, und somit werden keine Änderungen in anderen Bereichen benötigt, falls ein Dienstgeber geändert wird. Der Austausch von Dienstgebern ist selbst während der Laufzeit möglich.

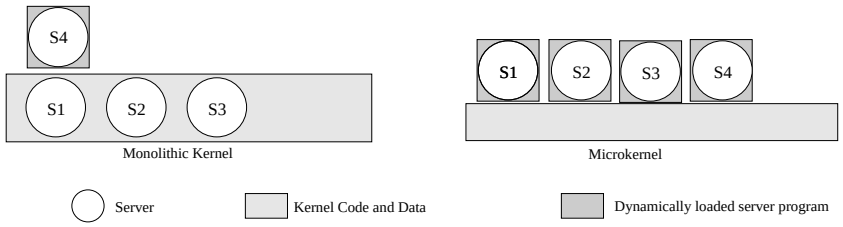

Abbildung 1.1: Vergleich von monolithischen und Mikrokern–Strukturen

Mikrokern–Betriebssysteme bieten die Möglichkeit, Programme unterschiedlicher Betriebssysteme nebenläufig ablaufen zu lassen. Diese Fähigkeit wird durch unterschiedliche Betriebssystempersonalitäten gewährleistet. Jedes Betriebssystem benötigt seine eigene Personalität, welche bei Systemaufrufen zur Abwicklung des Aufrufs herangezogen wird. Beim Start eines Programms werden Laufzeitbibliotheken in den Adreßbereich des Programms eingebunden. Diese Laufzeitbibliotheken garantieren zusammen mit den Personalitätendienstgebern denselben Ablauf eines Programms, als wenn es in seiner ursprünglichen Betriebssystemumgebung abgelaufen wäre.

Neue Computer–Architekturen, wie Multiprozessoren oder Multicomputer, stellen eine Herausforderung an die Betriebssystemarchitekten dar. Verteilte Systeme benötigen Unterstützung auf allen Ebenen des Systems. Benutzer wollen nicht mit der speziellen Hardware konfrontiert werden. Ihre Sicht des Systems sollte auf allen Architekturen dieselbe sein. Hierfür wird die Hardware in einer Schicht abstrahiert. Diese Schicht gewährleistet eine Einsystemsichtweise(*Single System Semantik*), auch wenn das System über mehrere Computer verteilt ist.

Die Dienste eines Systems sollten ausgewogen über mehrere Prozessoren verteilt sein. Diese Verteilung muß aber transparent für den Benutzer sein.

Er darf sich nicht darum kümmern müssen, auf welchen Knoten der Dienst angeboten wird. Anfragen an Dienste, die nicht auf dem eigenen Knoten angeboten werden, müssen vom System automatisch an die diensterbringenden Knoten weitergeleitet werden. Dort wird der Dienst erbracht und ggf. eine Rückantwort generiert.

Viele Mikrokerne bieten eine ortsunabhänige Interprozeßkommunikation, logische Adressierung und Namensgebung an, die eine transparente Verteilung von Kommunikationspforten und Prozessen erlaubt. Diese ortsunabhängige Interprozeßkommunikation ist die Grundvoraussetzung für eine transparente Dienstgeberverteilung.

Mikrokern–Betriebssysteme sind größtenteils plattformunabhängig programmiert, und die plattformabhängigen Teile sind in wenigen, gut strukturierten Modulen zusammengefaßt. Dies ermöglicht eine einfache und schnelle Portierung auf neue Hardwarearchitekturen.

Aufbau dieser Arbeit

Diese Arbeit faßt die wichtigsten Bestandteile der Thesis *„Microkernel Operating Systems in parallel Architectures“* zusammen. Hierfür wurden die wichtigsten Kapitel aufgegriffen und deren Ergebnisse ins Deutsche übersetzt. Es wird ein Überblick der bereits existierenden Mikrokerne und deren Entwurfsentscheidungen gegeben. Zusätzlich werden Pläne zu einem eigenen Mikrokern vorgestellt. Im folgenden Kapitel 2 werden die grundlegenden Eigenschaften eines Mikrokerns besprochen. Danach werden die Eigenschaften der Interprozeßkommunikation in Kapitel 3 vorgestellt. Die Schnittstelle zu existierenden Systemen und deren Integration in ein Mikrokern–Betriebssystem zeigt Kapitel 4, das die verschiedenen Lösungsansätze der Mikrokerne *Chorus* und *Mach* vergleicht. Eine Zusammenfassung der Eigenschaften des eigenen Mikrokerns *ODOS* wird in Kapitel 5 gegeben. Im abschließenden Kapitel 6 werden der Implementierungsstand und die geplanten Arbeiten an ODOS vorgestellt.

Kapitel 2

Die Architektur des Mikrokerns

Ein Mikrokern–Betriebssystem ist in zwei Schichten gegliedert. Die fundamentalen Dienste sind in der unteren Schicht, die Betriebssystem–spezifischen Dienste in der oberen Schicht implementiert. Die Dienste der unteren Schicht bilden den Betriebssystemkern, der Mikrokern genannt wird. Die Dienste der oberen Schicht können zwischen den verschiedenen Personalitäten variieren und werden Subsystemschicht genannt.

Durch die Modularität der Betriebssystemdienste können große Systeme in einer gut strukturierten Weise implementiert werden. Verschiedene Dienste können hierbei *als Zusatzmodule* installiert werden, die nur bei Bedarf geladen werden. Die Modularität erlaubt auch eine flexible Konfiguration zur Bereitstellung der gewünschten Funktionen.

Die meisten Dienste eines monolithischen Betriebssystems werden in der Mikrokernarchitektur im Subsystembereich implementiert. Das Subsystem benutzt die Dienste, die der Mikrokern anbietet, um dem Benutzer die gewohnte Betriebssystemumgebung zu gewährleisten. Ein Mikrokern kann hierbei mehrere Subsysteme zeitgleich ablaufen lassen. Bei entsprechender Programmierung können die jeweiligen Subsysteme sich gegenseitig Dienste anbieten. Ein besonders beliebter Dienst ist hierbei die Dateiverwaltung. Hierfür wird in ODOS die *Subsystem–unabhängige Schicht* eingeführt. Sie definiert allgemeine Dienstgeber, die von allen Subsystemen genutzt werden können. Diese allgemeinen Dienste sind in den meisten Systemen sehr ähnlich und müssen somit nicht mehr in jedem Subsytem implementiert werden. Nur die für eine neue Personalität spezifischen Dienstgeber müssen erstellt werden.

Prozesse verschiedener Personalitäten können nebenläufig ablaufen. Sie können Nachrichten aneinander schicken und somit auch Dienste anbieten und andere Dienste in Anspruch nehmen. Jeder Prozeß findet dieselbe Umgebung vor, wie wenn er auf einem System läuft, das von einem monolithischen Betriebssystem organisiert wird. Eine Personalität bietet somit ein *virtuelles Be-*

triebssystem. Systemaufrufe der Prozesse werden durch eine Bibliothek, die in den Adreßbereich des Prozesses eingebettet wird, auf die jeweilen Dienstgeber des Subsystems umgelenkt. Diese Umlenkung erfolgt ohne Kenntnisnahme des Prozesses.

Die Subsysteme können zusammenhängend in einem monolithischen Block oder auf mehrere Dienstgeber verteilt implementiert werden. Im verteilten Fall können die Dienstgeber auf mehreren Knoten verteilt werden. Das Ziel eines Mikrokerns sollte eine Verteilung auf mehrere Dienstgeber sein. Das *monolithische Subsystem* kann ein erster Schritt auf diesem Weg sein, da große Teile existierender monolithischer Systeme verwendet werden können.

Die Zweischichtenarchitektur bietet ein offenes Betriebssystem. Die Verteilung wird vom Kern organisiert und somit vor dem Benutzer verborgen. Dieses Konzept basiert auf dem Austausch von Nachrichten. Um die Verteilung der Anwendungen zu verbergen, adressiert Chorus alle Nachrichtenpforten mit einem systemweit gültigen Adresse, dem *Unique Identifier*. Der *Unique Identifier* adressiert Kommunikationspforten, die während ihrer Existenz auch auf andere Knoten wandern können. Durch Angabe eines *Unique Identifiers* als Adresse leitet das System die Nachricht an den richtigen Knoten weiter.

In den meisten Implementierungen ist ein Mikrokern verantwortlich für Interprozeßkommunikation, Ablauforganisation und Verwaltung des virtuellen Speichers und der physikalischen und logischen Ressourcen. Wie in Abbildung 2.1 gezeigt, ist der ODOS–Kern gut strukturiert und bietet folgende Dienste:

- ein hardwareabhängiger **Supervisor**, der externe Ereignisse abfängt und sie an die entsprechende Routinen weiterleitet;
- ein portierbarer **Instanzenverwalter**, der Instanzen verwaltet. Eine Instanz in ODOS ist verantwortlich für die Verwaltung von Betriebsmitteln wie Speicher und Pforten. Die Dienste, die dieser Instanzenverwalter anbietet, werden von Subsystemen erweitert, um die Semantik der vorgegebenen existierenden Betriebssysteme zu gewährleisten;
- ein portierbarer **Prozeßverwalter**(Threadmanager), der für prioritätsgesteuertes Ablaufplanung, Prozessorzuteilung und Prozeßerstellung verantwortlich ist. Prozesse können hierbei auf verschiedenen Prozessoren eines Multiprozessors ablaufen;
- ein teilweise portierbarer **Virtueller–Speicher–Verwalter**, der den lokalen und verteilten Speicher des Systems verwaltet.
- ein portierbarer **Interprozeßkommunikationsverwalter**, der asynchronen Nachrichtenaustausch und entfernte Prozeduraufrufe in einer ortsunabhängigen Weise anbietet.

In modernen Mikrokernen sollten keine Abhängigkeiten zwischen den einzelnen Modulen existieren. Die Schnittstellen jedes Moduls müssen klar definiert

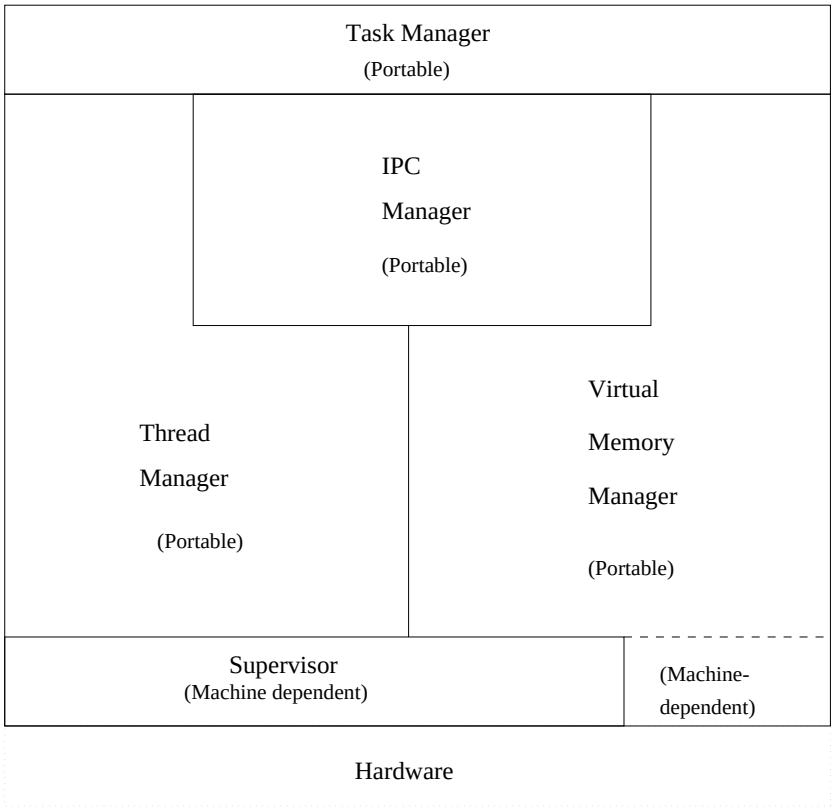

Abbildung 2.1: Mikrokern–Modularität

sein und von den anderen Modulen eingehalten werden. Somit ist eine Weiterentwicklung eines einzelnen Moduls möglich, ohne die komplette Funktionsweise der anderen Module zu kennen. Diese modulare Struktur ermöglicht eine schnelle und einfache Portierung auf andere Hardwarearchitekturen.

Ein weiterer wichtiger Punkt für den Erfolg eines Betriebssystems ist die einfache Programmierbarkeit. Eine standardisierte Programmierschnittstelle sollte für alle Hardwarearchitekturen eingehalten werden. Selbst Multiprozessoren sollten dieses Prinzip bewahren. Ein Programm mit mehreren Aktivitätsbahnen kann problemlos auch auf Einprozessorsystemen laufen. Das Betriebssystem muß hierbei die Organisation der einzelnen Aktivitätsbahnen übernehmen. Selbst als verteilte Anwendung auf einem Verbund von Rechnern sollte die Verteilung der Aktivitätsbahnen transparent für den Benutzer sein. Diese Transparenz kann durch eine einheitliche Adressierung der Objekte im kompletten System gewährleistet werden. Diese Adressierung erfolgt in ODOS durch den *eindeutigen Identifizierer*.

2.1 Eindeutiger Identifizierer

Die Adressierung aller Objekte erfolgt einheitlich über einen eindeutigen Identifizierer, dem *Unique Identifier* (UI). Im Gesamtsystem kann jeder UI nur einmal existieren. Er wird durch eine Kombination aus Erstellungsknotennummer und der Erstellungszeit erzeugt. Diese Idee des UI wurde von Chorus übernommen, wo jedes Objekt (actor, port, segment) einen UI besitzt. UI können nach der Erstellung auf einen anderen Knoten wandern, und das System leitet Nachrichten für diesen UI automatisch an den betreffenden Knoten weiter. Hier kann sehr einfach gewährleistet werden, daß der aktuelle, der Vorgänger und der Erstellungsknoten die aktuelle Position des UI kennen. Falls Nachrichten über Umwege an den Knoten geleitet werden, wird der sendende Knoten über den momentanen Aufenthaltsort des UI benachrichtigt. Indem diese drei Knoten den aktuellen Aufenthaltsort kennen, kann garantiert werden, daß jede Nachricht an den UI weitergeleitet wird.

Jeder Knoten unterhält eine Tabelle, in der die aktuellen Knoten der UI vermerkt sind. Diese Tabelle kann auf die UI beschränkt werden, die nicht auf ihrem Erstellungsknoten residieren. Zudem werden nur die UI vermerkt, zu denen in der letzten Zeit Nachrichtenkontakt bestand.

2.2 Wohlbekannte Pforten

Wohlbekannte Pforten (*Well–Known Ports*) sind Pforten, deren UI allgemein bekannt sind. Sie leisten allgemein nützliche Dienste, die nicht vom Kern übernommen werden. Es existiert eine Tabelle, in der die Dienste beschrieben sind und die diensterbringenden UI erfragt werden können. ODOS erweitert diese aus Chorus bekannte Methode um Fehlertoleranz. Falls ein Dienst angefragt wird, jedoch der Dienstgeber nicht aktiv ist, wird der Dienstgeber aktiviert, und er kann den angeforderten Dienst ausführen. Diese Methode stellt eine gewisse Fehlertoleranz bereit, da der Dienstgeber in Fehlerfällen wieder aktiviert wird. Zusätzliche Toleranz gegenüber Fehlern beinhaltet die Möglichkeit, daß wohlbekannte Pforten durch Pfortengruppen angesprochen werden können. Dies wird später erläutert.

2.3 Knoten

Die einzelnen Rechnerknoten sind in ODOS nicht nach außen als eigenständige Stationen zu erkennen. Das komplette ODOS–System wird als ein Rechner identifiziert. Es werden Dienste mit wohlbekannten Pforten bereitgestellt, die die Kommunikation zur Außenwelt sicherstellen.

Die Knoten innerhalb des Systems kennen die anderen Knoten im System und die Verbindungstopologie zwischen den Knoten. Diese Informationen werden für die Interprozeßkommunikation und den Lastenausgleich benötigt. Ein

Knoten ist hier als eine Gruppe eng gekoppelter Betriebsmitteln zu sehen, die von einem Mikrokern betrieben werden. Für die Kooperation zwischen den einzelnen Mikrokernen werden mehrere Pforten erstellt, die dem Statusinformationsaustausch zwischen den einzelnen Knoten dienen.

2.4 Instanzen und Aktivitätsbahnen

In modernen Betriebssystemen werden die Aktivitätsbahnen (threads) von den Instanzen (tasks) getrennt. Ein System kann hierbei mehrere Aktivitätsbahnen innerhalb einer Instanz verwalten, und es können mehrere Instanzen auf einem Knoten existieren. Damit werden nicht nur die mögliche Nebenläufigkeit, sondern auch die Betriebsmittel besser ausgenutzt.

Instanzen bieten hierbei eigene Adreßräume und einen Behälter für das Allokieren von Betriebsmitteln. In jeder Instanz laufen mehrere Aktivitätsbahnen, die die bereitgestellten Ressourcen verwenden. Hierfür müssen auch Möglichkeiten zum gegenseitigen Ausschluß zwischen den Aktivitätsbahnen bereitgestellt werden.

Im Gegensatz zu Chorus und Mach können Instanzen und Prozesse in ODOS wandern. Dies ermöglicht prinzipiell eine bessere Lastverteilung, jedoch müssen die Wanderkosten bei der Verteilung berücksichtigt werden. Aus diesem Grund wird die Möglichkeit der Migration nur in Ausnahmefällen ausgeschöpft werden. Wichtiger für eine gute Lastverteilung ist die Selektion des richtigen Ausführungsknotens bei Programmstart. Um hierbei eine gute Lastverteilung zu erhalten, kommunizieren die Mikrokerne und tauschen Information über ihre Prozessorlast aus. Instanzen sollten auch dort verwaltet werden, wo die Benutzung ihrer Betriebsmittel zu geringen unnötigen Kosten führen. Dies wird erreicht, indem kommunizierende Prozesse auf demselben Knoten oder bei hierarchischen Topologien auf nahen Prozessoren ausgeführt werden. Falls während der Ausführung die Kommunikationspartner wechseln, kann das System korrigierend eingreifen und den Prozeß zu seinen neuen Kommunikationspartnern verschieben. Hierbei muß jeweils der Nutzen mit den Kosten der Verschiebung abgewogen werden.

Prozesse sind an eine Instanz gebunden und können nicht außerhalb des von ihr bereitgestellten Adreßraums agieren. Sie können auf andere Knoten wandern, bleiben aber Teil der einen Instanz, dessen Betriebsmittel über mehrere Knoten verteilt sein können.

Instanzen stellen die Einheit der Betriebsmittelverwaltung dar. Betriebsmittel, die aus Initiative eines Prozesses innerhalb einer Instanz allokiert wurden, können von allen anderen Prozessen in der Instanz benutzt werden. Den Zugriff auf die Betriebsmittel regelt entweder das Betriebsmittel selbst, oder die Prozesse müssen sich untereinander koordinieren. Hierzu werden in der Instanz Sperren bereitgestellt, die die Prozesse zur Koordination benutzen können.

Kapitel 3

Interprozeßkommunikation

Die Interprozeßkommunikation in einem Mikrokern–Betriebssystem wird oft als Achillessehne bezeichnet. Der Grund hierfür liegt im großen Kommunikationsaufkommen, da viele Dienste nicht durch den Kern, sondern durch einen Dienstgeber bereitgestellt werden. Für die Kooperation zwischen den Dienstgebern und den Dienstnehmern werden keine Betriebssystemaufrufe, sondern Nachrichtentransfers benötigt.

Interprozeßkommunikation wird in ODOS ähnlich wie in Mach über Pforten abgewickelt. Diese Pforten sind an eine Instanz gebunden, und nur die Prozesse innerhalb dieser Instanz können Nachrichten über diese Pforte empfangen.

3.1 Sende– und Empfangsrecht

Um eine Nachricht über eine Pforte empfangen zu können, benötigt die Instanz des empfangenden Prozesses ein Empfangsrecht über diese Pforte. Pro Pforte gibt es genau eine Instanz, die das Empfangsrecht zu dieser hat. Das Empfangsrecht kann von einer Instanz zu einer anderen übergehen. Die darin gespeicherten Nachrichten bleiben dabei erhalten.

Um eine Nachricht an eine Pforte senden zu können, benötigt die Instanz des sendenden Prozesses ein Senderecht zu dieser Pforte. Senderechte an die Pforte können beliebig vergeben werden. Sobald eine Instanz ein Senderecht zu einer Pforte besitzt, können seine Prozesse Nachrichten an diese Pforte senden.

Es gibt verschiedene Senderechte. Als erstes kann eine Instanz ein Einmalsenderecht an eine Pforte haben. Dieses Senderecht erlischt, sobald eine Nachricht an diese Pforte versendet wird, oder die Instanz, die das Empfangsrecht hat, dieses Senderecht ihr wieder wegnimmt. Als zweites kann eine Instanz ein dauerndes Senderecht besitzen, das den Prozessen in dieser Instanz das Recht gibt, beliebig viele Nachrichten an die Pforte zu senden. Als drittes und letztes kann die empfangende Instanz an alle Tasks im System Senderechte geben.

Somit kann jeder Prozeß im System Nachrichten an die Pforte senden.

3.2 Pfortengruppen

Für einen Dienst kann es mehrere Instanzen geben, die Dienstanforderungen an dieselbe Adresse annehmen sollen. Dies ist nicht vereinbar mit der Eindeutigkeit des Empfangsrechts über eine Pforte. In ODOS gibt es hierfür Pfortengruppen, die eine eindeutige Adresse haben, aber deren Mitglieder sich dynamisch an– und abmelden können.

Diese Pfortengruppen dienen auch der Fehlertoleranz. Ein fehlertoleranter Dienstgeber wird durch einfaches Duplizieren der Instanz auf einem anderen Knoten erreicht. Eine Pfortengruppe kann angewiesen werden, die Nachrichten an alle Mitglieder weiterzusenden. Jedes der Gruppenmitglieder bearbeitet die Anfrage und stimmt das Ergebnis mit den anderen Gruppenmitgliedern ab. Danach übergeben sie die abgestimmte Version des Ergebnisses an den Dienstnehmer. Dies alles geschieht ohne Kenntnis des Dienstnehmers.

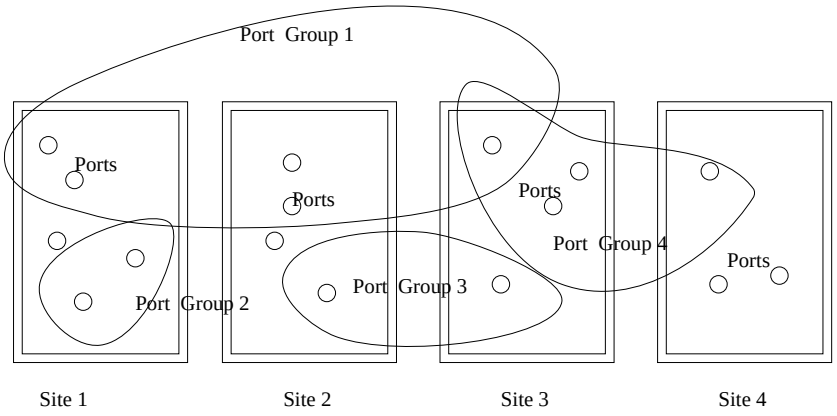

Abbildung 3.1: Pfortengruppen

Eine weitere Funktion dieser Pfortengruppen ist die Möglichkeit *Multicast*–Nachrichten zu verteilen, ohne daß der Sender unnötige Zeit im Senden und Überprüfen der Nachricht an jeden Partnerknoten verbringt. Jeder Partner muß als Mitglied in einer Pfortengruppe eingetragen sein und die Gruppe muß angewiesen sein, Nachrichten an alle Gruppenmitglieder zu verteilen. Somit ersparen sich Prozesse das Erstellen von mehreren Pforten und das Verwalten der Informationen welche Pforten genutzt werden können.

3.3 Kommunikation mit der Außenwelt

Ein ODOS–System zeigt sich nach außen als ein einzelner Rechner. Die einzelnen Knoten innerhalb des Systems sind nicht von außen adressierbar. Die

Kommunikation mit der Außenwelt findet über einen Protokolldienstgeber statt, der normalerweise auf dem Knoten installiert ist, der direkt mit der Außenwelt verbunden ist. Dieser wandelt ODOS–Nachrichten in Nachrichten eines speziellen Protokolls um und versendet sie an die Außenwelt.

Dieser Protokolldienstgeber erhält auch alle Nachrichten, die von außen an das System gesendet werden. Er wandelt die Nachrichten in ODOS–Nachrichten um und versucht, die Zielpforte der Nachricht zu ermitteln. Danach sendet er die Nachricht an diesen Zielpforte.

Somit sind alle Protokolle, die nicht der ODOS–IPC dienen, in einem Protokolldienstgeber implementiert und vom Kern extrahiert. Der zusätzliche Aufwand, der durch das doppelte Verschicken entsteht, ist nur dann bemerkbar, wenn der direkte Weg nicht über den Knoten des Protokolldienstgeber gegangen wäre. Dies ist jedoch nur dann der Fall, falls der Empfängerknoten an dasselbe Netz, wie das ODOS–System angeschlossen ist.

Kapitel 4

Unterstützung verschiedener Personalitäten

Die Akzeptanz eines neuen Betriebssystems ist ein wichtigster Faktor für den Erfolg und somit sehr wichtig im Entwurf. Akzeptanz kann erreicht werden, wenn das System überragende Funktionalität aufweist oder wenn es eine bessere Funktionalität bei Kompatibilität zu alten Systemen bietet. Der erste Punkt ist nur für Nischenprodukte zu erreichen. Der zweite Punkt ist jedoch auch bei allgemeinen Betriebssystemen zu erreichen, da neue Hardware neue Anwendungsgebiete öffnen und somit Funktionalitäten jetzt verwirklicht werden können, die beim Entwurf des Altsystems nicht denkbar waren.

Eine Personalität ist ein Subsystem oberhalb eines Mikrokerns, das dieselbe Funktionalität und die gleiche Programmierschnittstelle bietet wie das orginale Betriebssystem, das es emuliert. Als Personalitäten sind somit Emulationen aller herkömmlichen Betriebssysteme (Unix, Windows, VMS, MacOS) denkbar. Für die beiden Mikrokerne Mach und Chorus sind jeweils Unix–Personalitäten implementiert.

Mit dem Konzept der Personalitäten ist es auch denkbar, daß Programme verschiedener Betriebssysteme nebenläufig auf dem neuen Mikrokern–Betriebssystem ablaufen. Programme können somit ohne Änderung auf dem neuen System laufen.

Es existieren auch monolithische Betriebssysteme (z.B. OS/2), die andere Betriebssysteme emulieren. Diese Emulationen funktionieren nach einem anderen Prinzip. Die Systemaufrufe des emulierten Betriebssystems werden durch eine Laufzeitbibliothek in Systemaufrufe des emulierenden Betriebssystems umgeformt. Dies Umformung kostet unnötige Zeit und läßt das emulierte System nur als Untersystem des emulierenden Systems erscheinen.

Um Binärkompatibilität zu existierenden Betriebssystemen zu wahren, müssen folgende Anforderungen erfüllt werden:

- *Systemaufrufbearbeitung:* Alle Systemaufrufe müssen in derselben Weise behandelt werden, wie sie im ursprünglichen System behandelt werden.

- *Fehlersemantik:* Alle Fehler müssen in derselben Weise abgewiesen und ggf. eine Fehlermeldung an den Benutzer ausgegeben werden.
- *Schutz:* Benutzerdaten und Emulationsdaten dürfen nicht von der jeweils anderen Seite geändert werden können.
- *Signale:* Signale müssen in der selben Weise an die Benutzerprozesse weitergeleitet werden, wie das Orginalsystem es tut.

4.1 Implementierung von Personalitäten

Personalitäten sind als Subsysteme oberhalb des Mikrokerns implementiert. Diese Subsysteme müssen die Systemaufrufe des Benutzerprogramme verarbeiten, die normalerweise von dem Orginalbetriebssystem verarbeitet werden. Die Dienstgeber der Subsysteme sind als eingeständige Instanzen implementiert. Diese Instanzen werden im Benutzerbereich eines Systems ausgeführt. Um eine höhere Ausführungsgeschwindigkeit zu erreichen, können verschiedene Dienstgeber des Subsystems auch im Systembereich ablaufen(s. Abbildung 4.1).

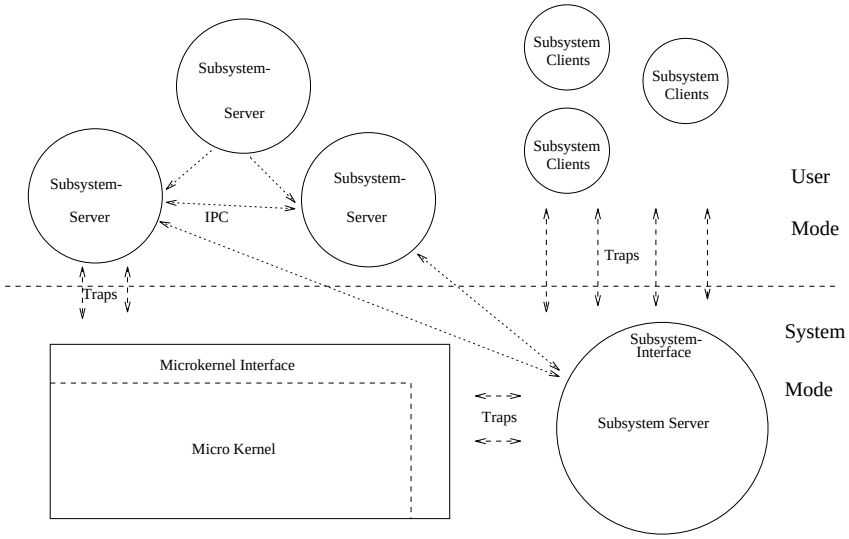

Abbildung 4.1: Integration eines Dienstgebers in den Systembereich

Ein Subsystem kann auf unterschiedliche Weise implementiert sein. CMU hat für Mach zuerst ein monolithisches BSD–Unix–Subsystem (s. Abschnitt 4.2) angeboten. Zur Zeit laufen Arbeiten, um ein Multiserver–Subsystem oberhalb von Mach zu verwirklichen. Für Chorus hat man ein System–V–Unix modularisiert und es als Multiserver–Subsystem(s. Abschnitt 4.3) oberhalb des Chorus–Kerns implementiert.

4.2 Monolithische Subsysteme

Während der Entwicklung von Mach hat man ein monolithisches BSD–Unix neu strukturiert und die wichtigen Dienste im Kern von den Unix–spezifischen getrennt. Diese Trennung ergab zwei Teile: den Mach–Mikrokern und eine monolithische BSD–Unix–Personalität. Wie diese Entstehungsgeschichte zeigt, können monolithische Personalitäten aus existierenden Systemen erstellt werden. Diese Erstellung der monolithischen Personalität kann aber nur ein erster Schritt in der weiteren Modularisierung der Betriebssystemdienste sein.

4.3 Multiserver–Subsysteme

Die Multiserver–Architektur ist ein Ergebnis eines gut strukturierten modularisierten Enwurfs. Verschiedene Dienstgeber sind getrennt in eigene unabhängige Instanzen, die durch Nachrichtentransfer miteinander kommunizieren. Chorus hat die Unix–Dienste von Anfang an neu strukturiert und in folgenden Modulen (s.Abbildung 4.2) implementiert:

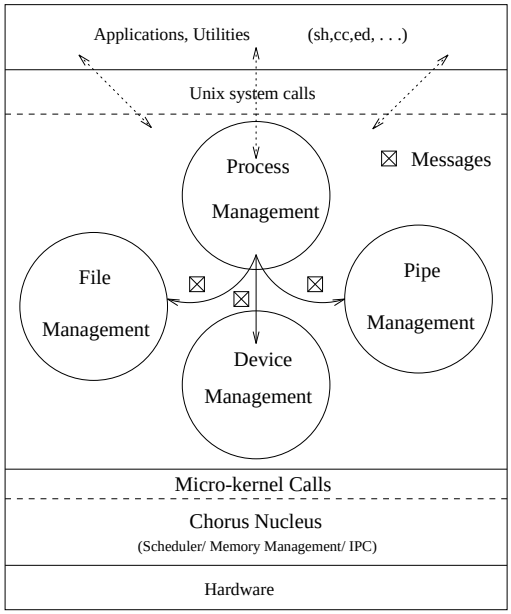

Abbildung 4.2: Modularer Aufbau von Chorus MiX

- *Prozeßverwalter:*
  Der Prozeßverwalter bildet Unix–Prozesse und Adreßräume auf die Chorus–Abstraktionen (cctors, threads und regions) ab. Er ist der zentrale Part in der Chorus/MiX–Implementierung und empfängt alle Systemaufrufe der Benutzerprogramme. Manche Dienste werden von ihm selbst erledigt, und andere leitet er über das Chorus–IPC an andere Dienstgeber

weiter. Durch die transparente Kommunikation in Chorus können die anderen Dienstgeber auch auf anderen Knoten residieren.

- *Dateiverwalter:*
  Der Dateiverwalter bietet den Prozessen ein Unix–Dateisystem. Er wird auf jedem Knoten, auf denen Platten installiert sind, ausgeführt. Plattenlose Knoten kommunizieren mit dem Dateiverwaltern auf anderen Knoten. Um die Dienste zu beschleunigen arbeitet der Dateiverwalter eng mit dem Verwalter des virtuellen Speichers zusammen.

- *Einheitentreiber:*
  Der Einheitentreiber bildet die Unix–*/dev*–Einheiten auf Chorus–Abstraktionen ab.

- *Socketverwalter:*
  Der Socketverwalter verwaltet die Netzwerkprotokolle wie TCP, UDP und IP, die über die BSD–Unix–Sockets angesprochen werden.

Ein großer Vorteil von Multiserver Subsystemen ist, daß ein Subsystem Dienste von anderen Subsystemen benutzen kann. Verschiedene Dienste können auf verschiedene Knoten verteilt sein. In Chorus ist dieses Konzept schon verwirklicht. Dateiverwaltungsdienste sind nur auf Knoten installiert, die auch Plattenanschluß haben. Einheitentreiber laufen nur dort, wo es Einheiten gibt, die verwaltet werden müssen. In Abbildung 4.3 wird dieser Entwurfsidee erläutert. Der mittlere Knoten ist an keine Einheiten angeschlossen. Somit benötigt man darauf nur einen Prozeßverwalter, der die Systemaufrufe aufnimmt und ggf. an Dienstanbieter auf anderen Knoten weiterleitet.

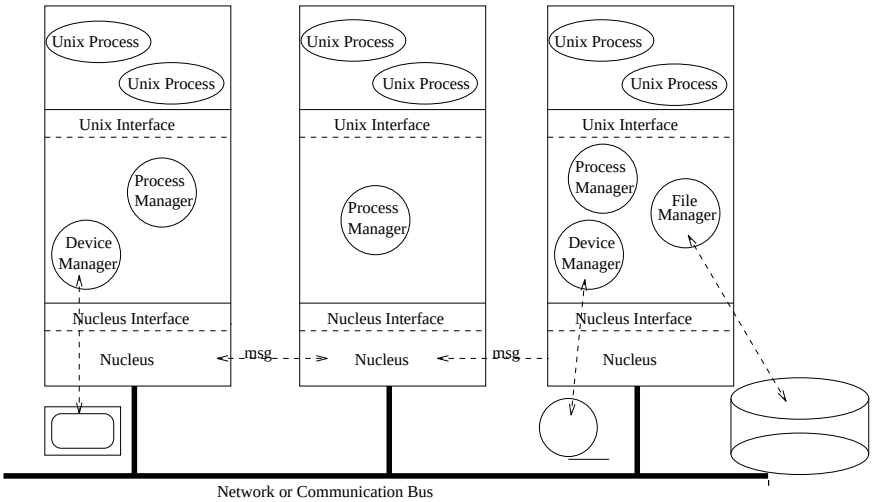

Abbildung 4.3: Verteilung von Dienstgebern über das gesamte System

Mach–US ist eine Multiserver–Implementierung oberhalb des Mach–Mikrokerns. Die Dienste sind ähnlich zu Chorus/MiX in einzelne Verwalter gegliedert.

4.4 Mehrere Personalitäten innerhalb eines Systems

Der große Vorteil eines Mikrokern–Betriebssystems ist, daß Programme verschiedener Betriebssysteme nebenläufig auf einem System ablauffähig sind. Jedes dieser Programme benötigt nur eine Menge von Dienstgebern (Personalitäten), die die eigenen Systemaufrufe bearbeiten. Die Kombination aus Laufzeitbibliothek, die bei Start des Programms in den Adreßraum gebunden wird, und der Menge von Dienstgebern ermöglichen es, daß der Mikrokern die System-aufrufe an die diensterbringende Seite weiterleitet. Abbildung 4.4 erläutert die Abgrenzung zwischen den einzelne Subsystemen.

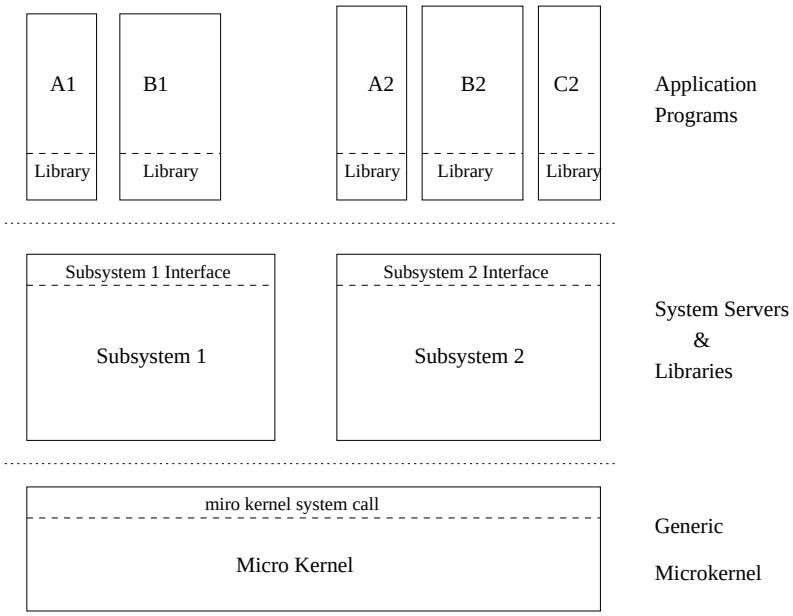

Abbildung 4.4: Mehrere Subsysteme in einem System

4.5 Personalitätsunabhängige Schicht

Große Teile verschiedener Betriebssysteme sind sehr ähnlich. Aus diesem Grund sollte ein Mikrokern–Betriebssystem allgemeine Dienstgeber anbieten, die von vielen Personalitäten genutzt werden. Ein Beispiel hierfür sind Netzwerkdienste und Dateiverwaltungsdienste. Dieses Konzept ermöglicht eine sehr schnelle und flexible Erstellung verschiedener Personalitäten, da große Teile nicht nochmal implementiert werden müssen. Personalitäten können auch Dienste an andere Personalitäten anbieten. Dieses gesamte Konzept ermöglicht eine sehr gute Kooperation zwischen den einzelnen Subsystemen. In Abbildung 4.5 sieht man, wie die verschiedenen Personalitäten die Dienste der unabhängigen Schicht in Anspruch nehmen. Es gibt jedoch auch Dienstgeber, die von einzel-

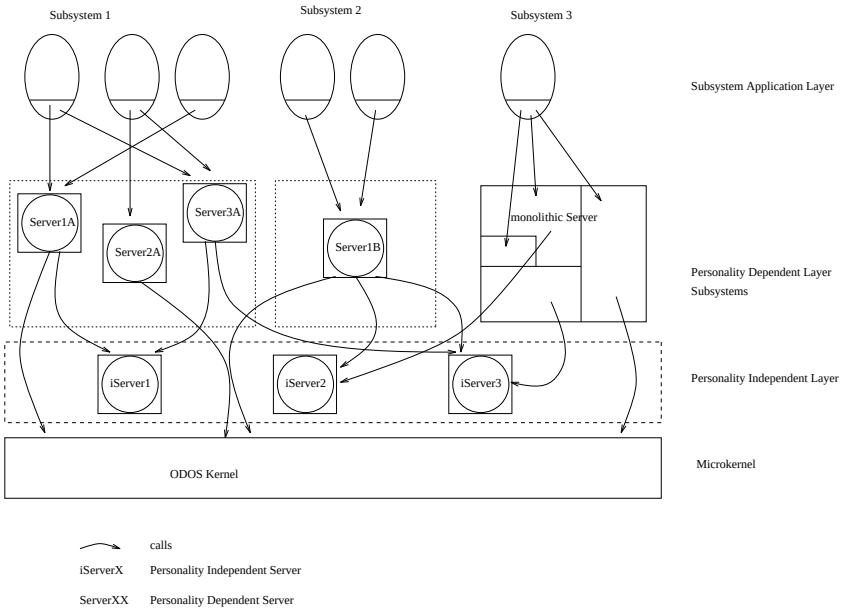

Abbildung 4.5: Personalitätenunabhängige Schicht

nen Personalitäten nicht benutzt werden. Dies ist der Fall, wenn die Dienste, die die Pesonalität anbietet, und die, die von der unabhängigen Schicht angeboten werden, sich zu sehr unterscheiden. In diesem Fall wird die benötigte Funktionalität in der Personalität selbst programmiert.

Kapitel 5

ODOS — ein neues System

Dieses Kapitel zeigt die wichtigsten Eigenschaften, die ein Mikrokern–Betriebssystem erfüllen sollte. Eine genaue Beschreibung der Elemente wird jeweils in des Kapiteln der *Thesis* gegeben.

5.1 Instanzen und Aktivitätsbahnen

- *multi–threaded:*
  Das ODOS–System ist ein Mehrinstanzensystem, das pro Instanz(task) mehrere Aktivitätsbahnen (threads) unterstützt. Jede Aktivitätsbahn wird in Form eines Prozesses eigenständig verwaltet und kann auf die Betriebsmittel, die seine Instanz bereitstellt, zugreifen. Durch die mehrfachen Aktivitätsbahnen innerhalb einer Instanz kann es zu echter Nebenläufigkeit kommen, indem verschiedene Prozesse auf verschiedenen Prozessoren ablaufen. Die initiale Verteilung der Prozesse wird durch das System, abhängig von der momentanen Lastverteilung, vorgenommen.

  Prozesse können während ihres Ablaufs auf verschiedenen Knoten ausgeführt werden. Innerhalb eines Knotens kann jeder Prozeß auf jedem beliebigen Prozessor ablaufen.

- *Lastausgleich:*
  ODOS versucht, die Last der einzelnen Knoten jeweils ausgewogen zu halten. Bei längerer unausgewogener Last kann das System korrigierend eingreifen und Prozesse von einem belasteten Knoten auf einen unbelasteten Knoten verschieben.

- *Ladezeit–Assemblierung:*
  ODOS bietet die Möglichkeit, Binärcode von Programmen in einer abstrakten Form zu speichern und beim Ladevorgang in die benötigte Zielbinärsprache zu übersetzen (siehe Taos–Mikrokern). Dadurch wird Red-

undanz in heterogenen Systemen vermieden und eine enge Kooperation zwischen zwei unterschiedlichen Rechnerarchitekturen ermöglicht.

- *Erweiterbarer virtueller gemeinsamer Speicher:*
  ODOS bietet für alle Instanzen einen über verschiedene Knoten verteilten virtuellen gemeinsamen Speicher an. Diese Unterstützung ermöglicht ein sehr einfaches Programmieren von parallelen Programmen, ohne komplizierten Datenaustausch selbst programmieren zu müssen. Dieser gemeinsame Speicher ist nötig, da Prozesse innerhalb einer Instanz auf verschiedenen Knoten ablaufen können. Realisiert wird dieser virtuelle gemeinsame Speicher durch eine Erweiterung eines von Mach bekannten *Memory Manager*, der entfernte Speicherseiten anfordert und gleichzeitige Lesezugriffe unterstützt.

- *Integration von Dienstgebern in den Systembereich:*
  ODOS unterstützt die Integration von Dienstgebern in den Systembereich eines ODOS–Systems. Diese Unterstützung ist nützlich, um:

  - Antwortzeiten von Dienstgebern zu reduzieren. Der Wechsel von Benutzer in den Systembereich und zurück kostet Zeit, die dadurch eliminiert werden kann. Insbesonders für Dienstgeber, deren Dienste sehr oft angefordert werden, ist diese Unterstützung sinnvoll.
  - manchen Prozessen direkten Hardwarezugriff zu erlauben. Hardware wird im Normalfall vom Kern abstrahiert und kann nicht direkt von Benutzerprozessen manipuliert werden. Alle Tasks im Systembereich können Hardware direkt benutzen und priviligierte Instruktionen ausführen. Diese Möglichkeit ist v. a. für Einheitentreiber nützlich.

5.2 Personalitäten

- *Binärkompatibilität:*
  Die Unterstützung verschiedener Betriebssysteme in Form von Personalitäten wird auf Binärcodebasis gewährleistet. Programme dieser Betriebssysteme können ohne Modifikation durch automatisches Hinzubinden einer Laufzeitbibliothek während des Programmstarts auf dem neuen System laufen. Die Personalitäten verarbeiten die Systemaufrufe der Programme und gewährleisten dieselbe Semantik wie das Orginalsystem.

- *Einzelsystem Semantik:*
  Auch für die Programme der Personalitäten erscheinen alle Knoten zusammen als ein Rechnersystem. Unabhängig davon, auf welchem Knoten sie ausgeführt werden, sind alle Betriebsmittel einheitlich nutzbar. Die Verteilung wird intern vom Kern verwaltet.

- *Personalitätsunabhängige Schicht:*
  Zur schnelleren Implementierung von Personalitäten werden standardisierte Dienstgeber angeboten. Jede Personalität kann Gebrauch von diesen Dienstgebern machen oder, falls die benötigte Funktionalität von der gebotenen abweicht, die Dienstgeber selbst implementieren.

5.3 Interprozeßkommunikation

- *Nachrichtentransfer:*
  Da das komplette System aus mehreren Knoten besteht und die Kommunikation zwischen allen Objekten im System einheitlich ist, werden alle Kommunikationen über Nachrichtentransport abgewickelt.

- *Sende–/ Empfangsrechte*
  Kommunikation erfolgt über den Nachrichtenaustausch an Pforten. Hierfür existieren Empfangsrechte, die das Empfangen von Nachrichten von einer Pforte erlauben, und Senderechte, die das Senden von Nachrichten an eine Pforte erlauben. Das Empfangsrecht über eine Pforte ist genau einer Instanz zugeordnet. Senderechte können von der Instanz, die das Empfangsrecht besitzt, an beliebige Kommunikationspartner vergeben werden. Es gibt verschiedene Arten von Senderechten.

- *Transparente Adressierung*
  Jegliche Adressierung im System ist ortsunabhängig. Egal ob Objekte auf dem lokalen oder auf einem entfernten Knoten adressiert werden, die Adressierung ist immer dieselbe.

- *Kommunikation mit der Außenwelt*
  Kommunikation mit der Außenwelt findet über spezialisierte Dienstgeber statt. Kommunikationsprotokolle sind aus dem Kern in einen Benutzerprozeß extrahiert und werden dort unterstützt.

- *LRPC und URPC*
  Optimierung für Rechnersysteme mit gemeinsamem Speicher wie *Lightweight– und User Level– Remote Procedure Calls* werden genutzt, falls die entsprechende Hardware vorhanden ist.

Kapitel 6

Schlußwort

Diese Zusammenfassung zeigte Gründe auf, warum Betriebssystemarchitekten neue Wege zur Realisierung komplexer werdender Systeme suchen mußten. Es wurden zwei Systeme vorgestellt und Verweise auf die genauere Beschreibung der Sachverhältnisse in der *Thesis* gemacht. Inbesondere wurde aufgezeigt, daß herkömmliche monolithische Systeme den Anforderungen, die parallele Architekturen stellen, nicht gerecht werden können.

Es wurde Lösungen in den Mikrokernen *Mach* und *Chorus* diskutiert und ihre Schwächen und Stärken erläutert. Für die Schwachstellen dieser beiden Systeme wurden Lösungsvorschläge in Form einer Beschreibung eines in Arbeit befindenden Systems gemacht. Die Ideen werden zur Zeit in einer Kommunikationsbibliothek verwirklicht. Diese Bibliothek eröffnet ein paralleles Programmieren auf einem lose gekoppelten Verbund von Arbeitsplatzrechnern. Die Ergebnisse, die bisher damit erreicht wurden, zeigen ein nahezu linearen Speedup und übertreffen somit die Erwartungen.

Auf diesem Rechnerverbund sollen alle Ideen, insbesondere die *Einsystem–Semantik*, implementiert und verfeinert werden. Die bisher erreichten Ergebnisse motivieren zu weiteren Schritten.

Als nächster Schritt wird ein *Abwickler* für den Rechnerverbund implementiert. Dieser Abwickler kooperiert mit dem vom System bereitgestellten Abwickler in der Weise, daß der neue Abwickler die parallelen Prozesse aktiviert und dem Unix–Abwickler zur weiteren Bearbeitung übergibt. Eine Erweiterung dieses Abwicklers wird dann in das System implantiert und statt des normalen Unix–Abwicklers verwendet.

Als weiterer Schritt wird auf einem Rechnerverbund ein virtueller gemeinsamer Speicher implementiert. Dieser virtuelle Speicher wird als *Memory Manager* des Mach–Mikrokerns realisiert. Parallele Programme, die bisher auf einem Parallelrechner mit gemeinsamen Speicher abliefen können somit auf den verteilten Rechnerverbund portiert werden. Vorarbeiten zu diesem Projekt laufen im Moment. Speicherverwaltung in verteilten Betriebssystemen ist eine große Herausforderung, und eine gute Implementierung ist abhängig von

einer leistungsstarken Kommunikationshardware zwischen den Knoten. Diese höchsten Kommunikationanforderungen genügende Hardware wird im Testsystem durch die ParaStation–[1]Einsteckkarte verwirklicht.

[1]ParaStation ist eine PCI–Einsteckkarte, die an der Universität Karlsruhe entwickelt wurde und speziell für die Parallelverarbeitung einen sehr hohen Durchsatz bei geringen Latenzzeiten bietet.