Diplomarbeit

Sicherheit und Performance langer Inter-Prozess Nachrichten

Tobias Bruns

30. Oktober 2006

Technische Universität Dresden
Fakultät Informatik
Institut für Systemarchitektur
Professur Betriebssysteme

Betreuender Hochschullehrer: Prof. Dr. rer. nat. Hermann Härtig
Betreuender Mitarbeiter: Dipl.-Inf. Marcus Völp

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Dresden, den 30. Oktober 2006

Tobias Bruns

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 L4 und die Mikrokernstruktur

Mikrokerne sind auf minimale Funktionalität getrimmte Betriebssysteme. Nur zwingend im Kernmodus ablaufende Funktionen wie die Verwaltung der Adressräume sind in diesen implementiert. Alle weiteren Funktionen des Betriebssystems werden von Servern übernommen, die alle in separaten Adressräumen laufen. Dadurch wird die Sicherheit und die Stabilität verbessert.

Die Server können untereinander und mit Nutzerprozessen über die Interprozesskommunikation, engl. Inter-Process-Communication (IPC), kommunizieren. Alle Programme müssen IPC verwenden, um Dienste eines Servers in Anspruch nehmen zu können. Daher ist die Performance der IPC-Nachrichten entscheidend für die Performance des gesamten Systems. Die so ausgetauschten Nachrichten können unterschiedlich groß sein. Sie können wenige Byte aber auch Megabyte an Daten transportieren.

L4 ist ein ursprünglich von Jochen Liedtke [Lie95] entwickelter Mikrokern. Fiasco [Hoh98, Hoh02] ist ein L4-kompatibler Kern, der von Michael Hohmuth und anderen entworfen und implementiert wurde. Der Fiasco-Kern bildet die Basis für diese Arbeit.

1.2 Ziel der Arbeit

Fiasco implementiert zur Übertragung von IPC zwei unterschiedliche Mechanismen, die beide durch den Kern durchgeführt werden. Kurze Nachrichten bis maximal 8 Byte werden direkt in Prozessorregistern übertragen. Längere Nachrichten werden durch den Kern vom Adressraum des Senders in den Adressraum des Empfängers kopiert. Dabei können Seitenfehler auftreten, die entsprechend behandelt werden müssen. Der dafür benötigte Verwaltungsaufwand fordert eine komplexe Zustandsmaschine.

Die eigentliche Nachrichtenübertragung wird unterbrochen und es wird eine weitere IPC-Nachricht aufgesetzt, um den verantwortlichen Pager über den Seitenfehler zu informieren. Währenddessen bleiben die kommunizierenden Threads im Zustand „IPC-In-Progress“, das heißt, dass eigentlich gerade eine Nachricht übertragen wird. Nachfolgende Tabelle zeigt die im Kern notwendigen Codezeilen, die zur Behandlung von langen IPC-Nachrichten notwendig sind.

Ziel meiner Arbeit ist es, alternative Mechanismen zur Übertragung von langen Nachrichten zu entwickeln und zu evaluieren. Dabei soll der Kern möglichst wenig beteiligt werden, um so die Codekomplexität verringern zu können. Dabei soll neben der Performance auch die Sicherheit berücksichtigt werden.

So müssen die Adressräume des Senders und des Empfängers getrennt bleiben. Keiner darf beliebige Daten aus dem fremden Adressraum auslesen oder verändern können.

Vorbereiten der Nachricht	58
Durchführen der Übertragung	42
Behandlung von Seitenfehlern	119
Gesamt	**219**

Tabelle 1.1: Codezeilen im Kern zur Übertragung von langen Nachrichten

Zusätzlich muss ein Server vor massenhaften Anfragen geschützt werden. Ein solcher Denial-of-Service-Angriff sollte die Funktionalität des Servers nicht beeinträchtigen.

Um trotz der Sicherheit eine hohe Performance zu erreichen, sollen die Vertrauensbeziehungen zwischen den Kommunikationspartnern untersucht werden. Ein hohes Vertrauen ermöglicht es, gewisse Sicherheitsmechanismen zu ignorieren um damit die Performance zu verbessern. Dies soll insbesondere systemnahe Prozesse begünstigen.

1.3 Größe von IPC-Nachrichten

Um sich mit langen Nachrichten beschäftigen zu können, soll zunächst ermittelt werden, wie lang in einem realem System vorkommende Nachrichten sind. Dadurch soll ermittelt werden, wie performancekritisch die Übertragung von langen Nachrichten ist, und wieviel Prozent der Nachrichten bereits durch kurze Nachrichten abgedeckt sind. Durch ein Ändern der Schnittstelle ist es möglich, längere Nachrichten als die bisherigen 8 Byte Nachrichten zu verschicken. So erlaubt die X2-Schnittstelle kurze Nachrichten bis 64 Worte.

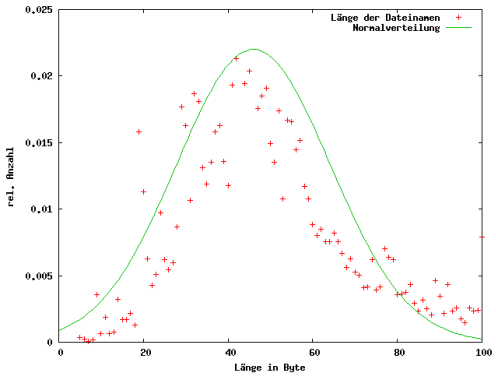

Abbildung 1.1: Verteilung der Länge von Dateinamen

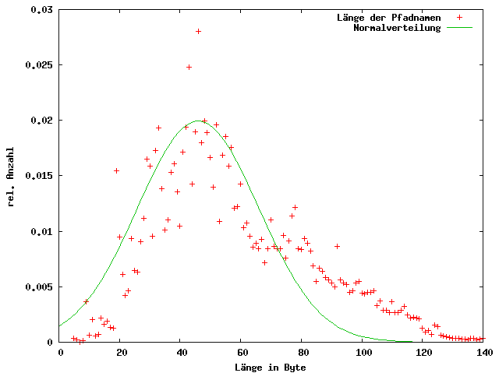

Abbildung 1.2: Verteilung der Länge von Pfadnamen

Die Paketgrößen können statisch oder dynamisch bestimmt werden. Bei der statischen Methode werden nur die beschriebenen Schnittstellen der einzelnen Pakete untersucht. Ein großer Nachteil dieser Methode ist, dass die unterschiedlichen Häufigkeiten der Nachrichten nicht berücksichtigt werden. Bei der dynamischen Methode werden in einem Testaufbau die tatsächlich auftretenden Nachrichten untersucht.

Für die statische Analyse wurde eine Auswahl an Paketen untersucht. Die berücksichtigten Pakete sind in Anhang A aufgeführt. Die Pakete definieren die Nachrichten, die sie empfangen und die dazu gehörende Antwort in einer Schnittstellen-Beschreibungssprache (IDL). Diese Sprache wird normalerweise durch einen Compiler, in diesem Fall DICE, in C übersetzt. DICE kann die Beschreibungen auch in XML übersetzen. Zur Auswertung dieser Daten wurde ein separates Programm entwickelt, das die Größe und die Anzahl der Nachrichten ermittelt.

Viele Nachrichten enthalten Strings dynamischer Länge. Daher kann hier die Größe nicht statisch angegeben werden und muss durch eine Funktion abgebildet werden. Bei diesen Strings handelt es sich in den ausgewählten Paketen um Dateinamen oder Pfadangaben. Die typische Länge dieser Dateinamen wurde durch Messung eines Referenzsystems bestimmt. Als Referenzsystem wurde eine Desktop Linux-Installation verwendet. Sie enthält neben den Betriebssystemkomponenten ein X-Windows und den Windowmanager KDE. Zusätzlich sind Nutzerdaten wie Dokumente und Multimedia-Dateien enthalten.

Die Länge der Nachrichten wurde durch eine Normalverteilung abgebildet, deren Streuung und Verteilung auf den ausgezählten Werten basiert. Die Abbildungen 1.1 und 1.2 zeigen, wie sich die gemessenen Längen verteilen und wie sich die verwendete Normalverteilungsfunktion dazu verhält.

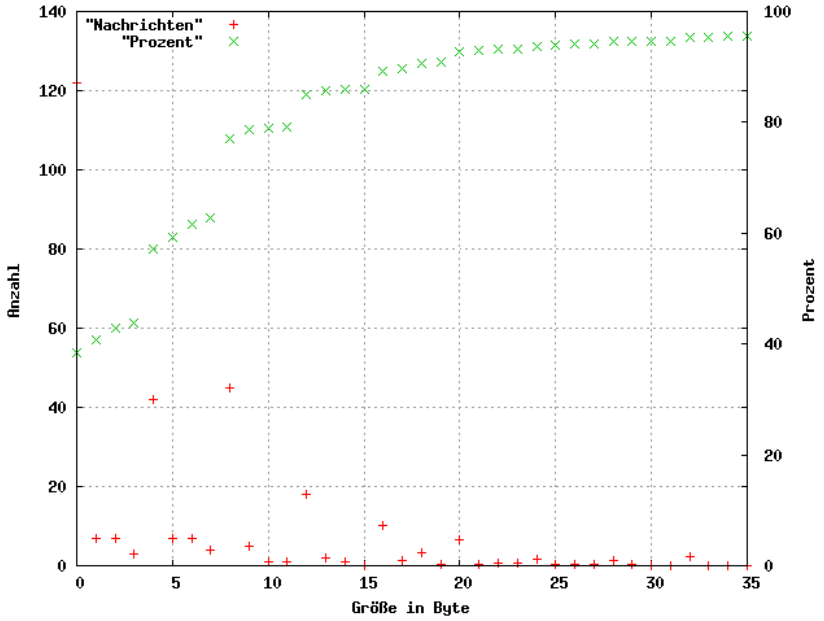

Abbildung 1.3: Nachrichtengröße, bestimmt durch statisches Auszählen

Die Ergebnisse der Auswertung sind in Abbildung 1.3 dargestellt. Sie zeigt die Menge der Nachrichten mit einer bestimmten Größe und den prozentualen Anteil der Nachrichten zu der Gesamtanzahl aller Nachrichten. Es zeigt sich, dass 95% aller Nachrichten kleiner als 30 Bytes sind. Bei dieser Bestimmung wurde aber nicht berücksichtigt, wie

oft eine Schnittstelle in einem realen System tatsächlich genutzt wird. Alle Nachrichten wurden gleich gewichtet.

Björn Döbel hat in seiner Diplomarbeit [Döb06] ein System entwickelt, mit dem die Länge der IPC-Nachrichten in einem laufenden System gemessen werden kann. Er führte eine Messung mit ORe, einem TCP-Stack, und eine Messung ohne ORe durch. Es zeigte sich bei der ersten Messung, dass etwa 75% aller Nachrichten kleiner als 30 Byte waren. Bei der zweiten Messung ohne Netzwerk waren sogar 98% aller Nachrichten kleiner als 30 Byte.

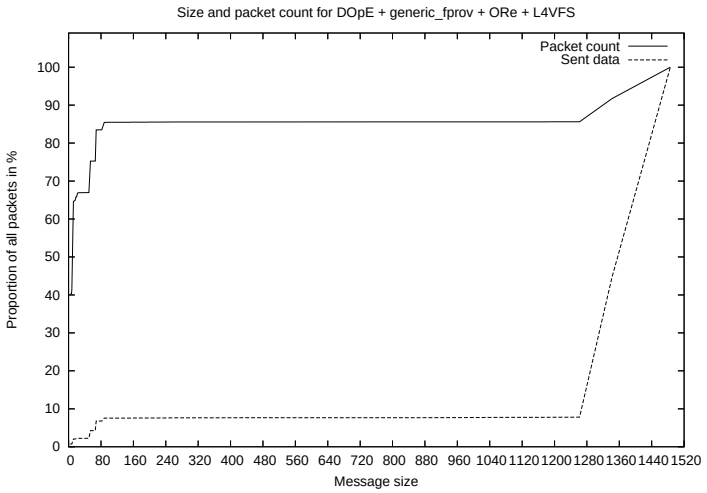

Abbildung 1.4: Nachrichtengröße nach dynamischer Messung aus [Döb06]

Die dynamische Messung zeigt, dass es Pakete gibt, in diesem Fall ORe, die viele sehr lange Nachrichten verwenden. Die jeweilige Zusammenstellung und Auslastung des Systems hat auf die Länge der Nachrichten einen großen Einfluss. Dennoch zeigten beide Verfahren deutlich, dass ein Großteil der Nachrichten kleiner als 30 Bytes sind. Zur Beschleunigung sollten solche Nachrichten als kurze Nachrichten behandelt werden.

In dieser Arbeiten sollen daher nur Nachrichten berücksichtigt werden, die größer als 30 Byte sind. Für solche Nachrichten sollen alternative Übertragungswege vorgestellt und nach verschiedenen Kriterien bewertet werden.

2 Stand der Technik

2.1 Inter Process Communication (IPC)

Threads tauschen Nachrichten über den Inter-Process-Communication-Mechanismus aus. In L4 gibt es Systemaufrufe zum einfachen Senden (*send*) und zum Empfangen von Nachrichten von einem beliebigen Sender (*wait*). Zum Empfangen von Nachrichten von einem bestimmten Sender (*closed-wait*) wird der Aufruf *receive* verwendet. Daneben gibt es noch Systemaufrufe für das kombinierte Senden und Empfangen (*replay_ and_ wait* und *call*). Die Nachrichtenübertragung ist synchron, das heißt sie findet erst statt, wenn Sender und Empfänger bereit sind. Um Blockierungen verhindern zu können, werden Timeouts gesetzt. Diese führen zu einem Abbruch der Verbindung, wenn ein angegebenes Zeitlimit überschritten wurde. Das Zeitlimit kann auch auf null gesetzt werden um ein generelles Blockieren zu verhindern. Für aufgetretene Seitenfehler können ebenfalls Zeitlimits gesetzt werden, die die maximale Bearbeitungszeit festlegen. Neben Daten können auch Flexpages, also die Rechte auf Speicherseiten, gesendet werden. Flexpages werden durch den Kern verarbeitet, der dann die Speicherseiten im Adressraum des Empfängers einblendet.

Als short-IPC werden Nachrichten bezeichnet, die eine bestimmte Größe nicht übersteigen. Bei L4.v2 liegt diese Grenze bei 8 Byte wobei die Daten dabei als Registerinhalte übertragen werden. Daher können keine Seitenfehler auftreten und die Übertragung ist sehr schnell. Die Nachricht kann statt der zwei Worte auch eine Flexpage enthalten.

Einen anderen Weg zur Übertragung ohne Seitenfehler ist die Verwendung des Userlevel Thread Control Block (UTCB). UTCBs sind Speicherobjekte, die vom Kern für jeden Thread angelegt werden. Der Kern sorgt dafür, dass sie stets eingeblendet sind, so dass keine Seitenfehler auftreten können. UTCB wurden in L4x2 neu eingeführt.

Längere Nachrichten werden als long-IPC bezeichnet. Sie können direkte Strings, indirekte Strings und mehrere Flexpages enthalten. Der genaue Aufbau der Nachricht wird im Nachrichtenkopf beschrieben. Die in einer Nachricht enthaltenen Flexpages werden vom Kern ausgewertet und die Rechte an den Speicherseiten dem Empfänger zugeschrieben. Die enthaltenen indirekten Strings werden durch den Kern in den Adressraum des Empfängers kopiert. Dazu wird der Teil des Empfängeradressraums mit dem Empfangspuffer in den Kernbereich des Senderadressraums eingeblendet. Nur der Kern hat Zugriffsrechte auf diesen Teil des Adressraumes, der auch IPC-Fenster genannt wird. Dann werden die Daten vom Senderadressraum kopiert.

Seitenfehler können dabei sowohl im Adressraum des Senders als auch im IPC-Fenster auftreten. Bei Seitenfehlern im Senderadressraum wird dieser an den Pager des Senders weitergeleitet. Der Nachrichtentransfer wird solange angehalten und die Parameter der Nachricht, Empfänger und Timeouts, werden gespeichert. Seitenfehler im IPC-Fenster werden vom Kern behandelt. Wurde der Seitenfehler ausgelöst weil die Seite im Empfän-

geradressraum nicht schreibbar ist, wird der Seitenfehler an den Pager des Empfängers weitergeleitet.

2.2 Die L4-Schnittstellenbeschreibungen

v2 ist die in [Lie95] beschriebene Schnittstelle und ist die bisherige stabile Version. Sie bildet auch die Grundlage für meine Implementierung. Kurze IPC-Nachrichten umfassen maximal 8 Bytes und werden als Registerwert übertragen.

x.2 ist eine experimentelle Version, die in die stabile Version v.4 übergehen soll. Die für diese Arbeite relevanteste Änderung ist die Einführung von UTCBs zur Vergrößerung von kurzen Nachrichten. Sie ist bereits Pistachio-Kernel implementiert wo kurze Nachrichten bis zu 64 Datenworte enthalten können. Auch für den Fiasco-Kern wurden die x.2 Schnittstellen implementiert [Cla04].

L4.sec ist eine Version, die insbesondere Rechte- und Ressourcenprobleme behandelt [KV05]. In L4.sec können Rechte an Ressourcen in Form von Capabilities übertragen werden. Diese Capabilities können auch Zugriffsrechte auf Speicherinhalte im Adressraum eines anderen Prozesses enthalten. Dies kann zur Datenübertragung verwendet werden, in dem das Zugriffsrecht auf den Sendepuffer übertragen wird. Das Zugriffsrecht kann bei fehlverhalten sofort wieder entzogen werden. Für der Fiasco-Kern wurde auf L4.sec [Kau05] portiert.

2.3 Verwandte Arbeiten

2.3.1 Mach Ports

Mach ist ein Mikrokern [Loe92b, Loe92a], der im Rahmen eines Projekts an der Carnegie-Mellon Universität entwickelt wurde. Für die Inter-Prozess-Kommunikation kommen Nachrichten und Ports zum Einsatz. Die Ports sind unidirektionale Schnittstellen, über die die Prozesse Nachrichten empfangen. Um eine Nachricht an einen solchen Port schicken zu dürfen, muss ein Prozess erst das Recht dazu erhalten haben. Die Rechte werden ebenfalls als Nachrichten verschickt. Ein neu angelegter Prozess erhält das Recht, dem bootstrap-Thread Nachrichten zu schicken. Dieses wird dann genutzt, um anderen Prozessen Rechte zukommen zu lassen. Die Ports enthalten auch eine FIFO-Warteschlange für die eintreffenden Nachrichten. Bis die Warteschlange voll ist, können Nachrichten asynchron abgearbeitet werden.

Auch in Mac OS X [App06] kommen Mach Ports für die interne Kommunikation zum Einsatz.

2.3.2 Lightwight Remote Procedure Call

Lightwight Remote Procedure Call (LRPC [BALL90]) ist eine schnellere Variante als die herkömmlichen Remote Procedure Calls. LRPC wurde entwickelt und optimiert für

die Kommunikation zwischen Sicherheitsdomains auf der gleichen Maschine. Es bietet dabei sowohl Sicherheit als auch Performance indem es die Overhead reduziert. LRPC wurde in der DEC SRC Firefly Workstation eingebaut und ist dreimal schneller als die herkömmlichen RPCs [SB90].

2.3.3 Trusted Buffer Object

In [Sha03] stellt Jonathan S. Shapiro den IPC-Mechanismus in dem EROS-Betriebssystem [Sha99] vor. Die Übertragung von einem langen Datenstring wird über einen Zwischenspeicher, dem sogenannten Trusted Buffer Object, durchgeführt. Die Implementierung des Zwischenspeicheres ist unabhängig vom Client und somit für den sendenden Server vertrauenswürdig. EROS verfügt über Mechanismen, über die der Server sich von der Autentizität des Puffers überzeugen kann. Dann werden die Daten zunächst in den Puffer übertragen, für den der Client Speicher zur Verfügung stellt. Damit ist die Übertragung für den Server abgewickelt.

2.3.4 Raven Kern

Der Raven Kern [Rit93] ist ein leichtgewichtiger Prozessorkern, der viele betriebssystemtypische Dienste in den User-Space auslagert. Der Kern selbst enthält noch die Taskverwaltung, das Speichermanagement und Mechanismen zur Ausnahmenbehandlung. Die Inter-Prozess-Kommunikation kommt im Idealfall ohne Kerninteraktion aus [RN93]. Die IPC-Bibliothek stellt ein portbasiertes asynchrones und synchrones Send- und Empfangsinterface zur Verfügung. Zur Übertragung von großen Datenmengen wird der Speicherbereich mit der Nachricht im Empfängeradressraum eingeblendet.

3 Entwurf

3.1 Anforderungen

3.1.1 Performance

Die gesamte Kommunikation zwischen den Betriebssystemkomponenten läuft über IPC-Nachrichten. Daher wirkt sich die Performance des IPC-Systems direkt auf die Performance des gesamten Betriebssystems aus.

Häufig werden IPC-Nachrichten zum Signalisieren von Zuständen verwendet. Bei solchen Nachrichten ist die Latenz wichtiger als die Bandbreite, da diese nur sehr geringe Datenmengen enthalten aber möglichst schnell zugestellt werden sollen. Wenn tatsächlich größere Datenmengen übertragen werden müssen, spielt die Latenz neben der Bandbreite nur eine untergeordnete Rolle. Wichtiger ist dann die zügige und sichere Übertragung der Daten.

3.1.2 Isolation

Zur Isolation der Speicherbereiche von Threads erhält jeder Thread einen eigenen Adressraum. Dieser stellt virtuellen Speicher dar, der auf den realen Speicher abgebildet wird. Der Zugriff auf die Speicherbereiche ist nur nach einer expliziten Authorisierung erlaubt. Die Zugriffsrechte werden in Lese- und Schreibrechte unterteilt. Der Speicher wird in Seiten aufgeteilt, die bei der Intel-Architektur 4 Kilobyte groß sind. Die Rechte werden seitenweise gewährt.

Bei einer Übermittlung von Daten über gemeinsamen Speicher muss die Granularität der Zugriffsrecht bei der Platzierung berücksichtigt werden. Wenn sie nicht genau am Anfang einer Seite beginnen und am Ende einer Seite aufhören, kommt es zu einem Verschnitt. In diesem Bereich können sich vertrauliche Daten befinden. Um die Zugriffsrechte an einem Puffer korrekt verteilen zu können, muss ein Verschnitt verhindert oder nachträglich durch Kopieren des Puffers aufgelöst werden.

3.1.3 Denial of Service

In [LIJ97] werden die verschiedene Formen von Denial-of-Service-Angriffen und Schutzmaßnahmen für Server diskutiert. Es werden Pulsar-Angriffe und endloses Senden oder Empfangen als Angriff unterschieden.

Bei einem Pulsar sendet der Angreifer so schnell wie möglich unsinnige Nachrichten. Der Empfänger muss diese Nachrichten empfangen und als unsinnig identifizieren. In dieser Zeit können keine anderen Nachrichten empfangen werden.

Ein endloses Senden wird erreicht, indem der Sendepuffer nicht gemappt ist. Dies löst beim Übertragungsversuch einen Seitenfehler aus, der vom Pager des Senders behandelt werden müsste. Geschieht dies nicht, wird auch der Empfänger blockiert.

Bei nicht kernbasierten Übertragungsmechanismen sind zusätzliche Protokollnachrichten notwendig. Bleiben diese aus oder werden nicht korrekt beantwortet, stellt dies eine weitere Form des Denial-of-Service-Angriffs dar.

3.1.4 Übertragungsmodi

Der einfachste Übertragungsmodus ist Unicast, das heißt, die Nachricht wird von einem Empfänger genau einem Absender zugestellt. Dabei wird noch unterschieden, ob die Nachricht synchron oder asynchron übertragen wird. Bei der synchronen Übertragung wird die Nachricht direkt vom Sender zum Empfänger zugestellt. Dazu muss entweder der Sender auf die Empfangsbereitschaft des Empfängers warten oder der Empfänger muss auf eine Nachricht warten. Bei der asynchronen Übertragung wird die Nachricht abgesendet und wenn sie nicht direkt zugestellt werden kann, wird sie zwischengespeichert.

Eventuell muss eine Nachricht an mehrere (Multicast) oder sogar an alle (Broadcast) Server in einem System geschickt werden. Der Sender könnte die Nachricht mehrmals abschicken, allerdings ist dies teurer als die Nachricht bei der Zustellung an mehrere Empfänger zuzustellen. Wird die Nachricht nur einmal abgeschickt und an mehrer Empfänger zugestellt, muss die Initialisierung des Sendepuffers nur einmal vorgenommen werden. Allerdings benötigt dieser besondere Modus ein eigenen Aufbau zur Verarbeitung und Anzeige von Zustellungsfehlern.

Nachrichten verlangen nach einem gewissen Grad an Transaktionssicherheit. Alle Nachrichten müssen vollständig und unverändert oder, wenn dies nicht gewährleistet werden kann, gar nicht zugestellt werden. Eine unvollständige Nachricht kann so zu falschen Interpretationen auf der Empfängerseite führen und somit zu falschen Reaktionen. In bestimmten Fällen dürfen Nachrichten nicht mehrmals zugestellt werden. Der Empfänger kann sonst ebenfalls falsch reagieren, da auf jeden Nachricht einzeln reagiert wird.

3.1.5 Vertrauensbeziehungen

Sicherheitsbedingungen können zu Gunsten von Performancesteigerungen eingeschränkt werden, wenn zwischen den Kommunikationspartnern bereits eine Vertrauensbeziehung besteht. Dann soll davon ausgegangen werden, dass es zu keinem mutwilligen Denial-of-Service-Angriff kommt. Die Isolation der Prozesse soll dennoch erhalten bleiben. Insbesondere muss noch ein Schutz vor fehlerhaftem Verhalten, das durch Fehler im Programm verursacht wird, bestehen bleiben.

Bei systemnahen Servern kann man generell von einer höheren Stabilität und Vertraulichkeit ausgehen als zum Beispiel bei einem Nutzerprozess. So können Systemprozesse untereinander mit vollständiger Vertrauensvorbedingung miteinander kommunizieren. Ein Benutzerprozess kann auch einem Systemprozess volles Vertrauen entgegen bringen, umgekehrt wird dies aber nicht erwidert. So entstehen vier Vertrauenskonstellationen:

- unidirektional durch den Empfänger: Der Empfänger vertraut dem Sender, umgekehrt aber nicht. Diese Konstellation tritt auf, wenn ein Server eine Antwort an den Client sendet.
- unidirektional durch den Sender: Der Sender vertraut dem Empfänger, umgekehrt aber nicht. Diese Konstellation tritt beim Senden einer Clientanfrage an den Server auf.
- bidirektional: Empfänger und Sender vertrauen sich gegenseitig. Dies ist häufig bei der Kommunikation zwischen zwei Systemprozessen zu finden.
- kein Vertrauen: Empfänger und Sender vertrauen sich nicht. Diese ist für die Kommunikation zwischen Benutzerprozessen typisch.

Das Wissen um diese Vertrauensbeziehungen soll dazu genutzt werden, um bei der Wahl des Übertragungsmechanismus die Performance zu steigern.

3.2 Direkte Kommunikation über kurze Nachrichten

Es sollen verschiedene Methoden zur Übertragung von Nachrichten zwischen zwei Prozessen mit getrennten Adressräumen betrachtet werden. Da dabei Nachrichten betrachtet werden, die länger als short IPC sind, kommt die Übertragung mittels einer kurzen Nachricht nicht in Frage.

Alternativ kann man die Nachricht auf mehrere Nachrichten aufteilen und diese als Sequenz von kurzen Nachrichten verschicken. Der Empfänger setzt dann aus den einzelnen Nachrichten die gesamte Nachricht zusammen. Dazu muss der Sender die Menge der zu erwarteten Nachrichten mitteilen. Bei sehr langen Nachrichten muss die Sequenz viele Nachrichten enthalten. Während der Übertragung der Sequenz müssen Nachrichten, die nicht zu der Sequenz gehören, erkannt und gesondert verarbeitet oder verworfen werden. Möglich ist zum Beispiel ein closed-wait um auf die Nachrichten der Sequenz zu warten. So werden nur Nachrichten vom Sender der Sequenz empfangen. Dies blockiert allerdings den Empfänger, wenn der Sender die Sequenz nicht korrekt abschließt. Der Empfänger muss die Sequenz nach Erreichen eines Zeitlimits verwerfen um wieder andere Nachrichten empfangen zu können.

Die Übertragung über eine Sequenz von kurzen Nachrichten hat eine sehr gute Latenz, da durch den geringen Übertragungsaufwand die ersten Daten sehr schnell übertragen werden können. Allerdings ist die Bandbreite stark begrenzt, da pro Nachricht nur wenige Daten übertragen werden können und durch die häufigen Kontextwechsel sehr hohe Kosten entstehen. Auf dem in der Evaluation verwendeten Rechner (siehe Kapitel 5) kostet eine kurze Nachricht etwa 2000 Takte und eine lange Nachricht mit 8 Datenworten kostet etwa 5300 Takte. Für die 8 Worte braucht man in L4v2 aber schon 4 kurze Nachrichten, was deutlich teurer ist.

3.3 Direkte Kommunikation über geteilten Speicher

Eine preiswerte Methode ist die Übertragung über einen geteilten Speicherbereich. Dabei erhalten die Kommunikationspartner Lese- bzw. Schreibrechte auf einige Speicherseiten. In diese schreibt der Sender die Nachricht und der Empfänger kann sie auslesen. Der Sender muss dabei darauf achten, dass sich keine weiteren Daten aus seinem Adressbereich indem gemeinsam genutzten Adressbereich befinden, da sonst der Empfänger auch diese Daten auslesen kann. Dies kann erreicht werden, in dem die Nachricht vor dem Versenden in den geteielten Speicher kopiert wird (Copy-In). Der Empfänger kann die Nachricht zum raschen Freigeben des Speicheres herauskopieren (Copy-Out). Generell kann dieses Verfahren aber auch ohne jede Kopieroperation durchgeführt werden. Dann baut der Sender die Nachricht direkt in diesem Speicherbereich auf und der Empfänger liest sie aus.

Beim Schreiben oder Auslesen der Nachricht können Seitenfehler auftreten. Diese müssen von einem Pager aufgelöst werden. Als Pager kann einer der Kommunikationspartner dienen oder ein neutraler Prozess. Ein Regionmapper wird dann dazu benötigt, um die im geteiliten Speicher auftretenden Seitenfehler dem zuständigen Pager zuzuweisen.

Die im Seitenfehler angegebene Adresse bezieht sich auf die Seite im Adressraum des betroffenen Prozesses. Die Seite kann im Pager in einer anderen Adresse liegen. Die Adressen der betroffenen Seiten müssen übersetzt werden. Wenn ein Regionmanager verwendet wird, übernimmt dieser die Übersetzung der Adressen. Ansonsten muss es der Pager selbst vornehmen. Dazu sind aber Informationen über den Aufbau des Adressraumes des anderen notwendig.

Die beiden kommunizierenden Prozesse müssen dem auserwählten Prozess vertrauen können. Die Seitenfehler müssen zuverlässig aufgelöst werden und wenn ein dritter Prozess als Pager fungiert, darf dieser die Nachrichten nicht abhören. Wird ein Seitenfehler nicht innerhalb einer bestimmten Zeit behandelt, wird der Vorgang abgebrochen. Die Übertragung der Nachricht kann dann nicht abgeschlossen werden. Seitenfehler können auch zum sofortigen Abbruch der Kommunikation führen. Dies bietet sich an, wenn eine Nachricht schnell übertragen werden muss. Dann ist es unter Umständen besser, sie wird nicht übertragen als verzögert übertragen, da die Behandlung eines Seitenfehlers sehr teuer sein kann.

Um Speicherseiten im Adressraum eines anderen Prozesses einzublenden wird eine Flexpage verschickt. Dieser Vorgang ist recht teuer. Daher ist es günstiger, den einmal geteilten Speicher auch für zukünftige Nachrichten zu verwenden. Dann muss der Sender aber immer ein Copy-In durchführen oder er verwendet den geteilten Speicher als Nachrichtenpuffer, in dem er die Nachricht aufbaut. Dieses Vorgehen bietet sich besonders an, wenn häufig lange Nachrichten verschickt werden müssen und Empfänger und Sender gleich bleiben.

Die Übertragung über einen geteilten Speicher weist zwei Probleme auf. Zum einen die Signaliserung, mit der angezeigt wird, wann neue Nachrichten beim Empfänger angekommen sind und umgekehrt wann Nachrichten empfangen wurden. Außerdem dürfen bereits gesendete Daten nachträglich nicht vom Sender verändert werden oder durch die Übertragung verfälscht werden. Eine Nachricht muss vollständig oder gar nicht zugestellt

werden. Da der Sender aber Schreibrechte auf das gemeinsame Speichersegment hat, müssen geeignete Mechanismen dafür sorgen, dass der Sender zumindest nicht unbeabsichtigt bereits gesendete Nachrichten überschreibt. Dazu kann der Empfänger die Nachricht aus dem geteilten Speicher kopieren und den Nachrichtenspeicher sofort wieder freigeben. Dies ist billiger als für jede Nachricht neuen Speicher zu verwenden und die Rechte an diesem zu übertragen.

Der geteilte Speicher kann auch als Ringpuffer genutzt werden, bei dem die zu übertragenden Nachrichten hintereinander geschrieben werden. Die Signalisierung kann dabei auf ein Minimum reduziert werden. So müssen neue Nachrichten nur angezeigt werden, wenn der Puffer zuvor leergelaufen ist. Die Übertragung erfolgt so asynchron und der Sender bekommt nicht sofort eine Bestätigung, dass die Nachricht zugestellt werden konnte.

Ein Streaming-Interface zur echtzeitfähigen Übertragung von Daten über einen geteilten Speicher ist in [LRH01] vorgestellt. Hier werden auch die notwendigen Signalisierungsmechanismen diskutiert.

3.4 Indirekte Kommunikation über einen zentralen Server

Bei der Übertragung über einen Server wird die Nachricht zuerst vom Sender zum Server und dann vom Server zum Empfänger übertragen. Die Kommunikationspartner müssen dem Server vertrauen, aber umgekehrt braucht der Server den Clients nicht zu vertrauen und diese sich untereinander auch nicht.

Es werden also zwei getrennte Nachrichtenaustausche betrachtet, die als direkte Kommunikation mit einseitigem Vertrauen realisiert werden. Die Vertrauensverhältnisse sind bei den beiden Kommunikationsvorgänge verschieden, da der Server einmal die Rolle des Senders und einmal die des Empfängers innehat. Bei der Übertragung vom Senderclient zum Server vertraut der Sender dem Empfänger. Im nächsten Schritt wird die Nachricht vom Server zum Empfängerclient gesendet, wobei der Empfänger dem Sender vertraut.

Der Server stellt die vermittelnde und schützende Instanz dar, die bei der ursprünglichen Übertragung der Kern dargestellt hat. Der Server sollte den Clients Schutz vor Angriffen bieten, und muss das Einhalten von Restriktionen und Timeouts überwachen. Diese Sicherheitsbedingungen müssen beim Design des Übertragungsprotokolls für die Kommunikation zwischen den Clients und dem Server berücksichtigt werden. Die Tatsache, dass die Clients dem Server vertrauen, kann allerdings dazu genutzt werden, um einige Sicherheitsbedingungen zu Gunsten einer besseren Performance unberücksichtigt lassen zu können.

Die fehlerhafte Abarbeitung des Kommunikationsprotokolls auf der Clientseite darf die Betriebsbereitschaft des Servers nicht beeinträchtigen. So sollte der Server nicht blockieren, während er auf Protokollnachrichten oder auf die Behandlung eines Seitenfehlers wartet. Wird dann ein Zeitlimit überschritten, wird eine Fehlermeldung an die Clients geschickt und die Übertragung wird abgebrochen.

Ein Denial-of-Service-Angriff durch Senden von vielen Nachrichten kann nicht direkt den Empfänger treffen, da alle Nachrichten zunächst den Server passieren müssen. Dieser hat dann die Möglichkeit, einen solchen Angriff zu erkennen und die Kommunikation zu

einem fehlerhaft arbeitenden Client abzubrechen oder zu verlangsamen, so dass keine Gefahr für den weiteren Betrieb besteht.

3.4.1 Copy Server

Bei diesem Ansatz werden die Speicherseiten von den Clients in den Adressraum des Servers eingeblendet und dieser kopiert daraufhin die Datenworte von der Speicherseite des Senders in die Speicherseiten des Empfängers. Nach Abschluss des Kopiervorgangs bekommen die Clients eine entsprechende Nachricht. Seitenfehler treten nur im Adressraum des Servers auf. Dieser muss sie einem Client zuordnen, der dann für die Behandlung verantwortlich ist.

Die Clients müssen dem Server vertrauen, dass dieser nur die zu versendenden Daten kopiert und andere Informationen auf den Speicherseiten nicht ausliest oder manipuliert. Andernfalls müssen sie selbst dafür sorgen, dass sich keine vertraulichen Daten auf diesen Speicherseiten befinden.

Eine asynchrone Übertragung ist nur realisierbar, wenn ein Thread in den Clients Seitenfehler des Servers beantwortet. Dann kann der bereits gemappte Speicher als Ringpuffer verwendet werden, der mehrere Nachrichten aufnimmt. Die jeweiligen Empfänger könnten dann im Nachrichtenkopf vermerkt sein. Nachfolgend soll nur die synchrone Übertragung detailliert betrachtet werden, das Übertragungsprotokoll ist aber auch für die asynchrone Übertragung verwendbar.

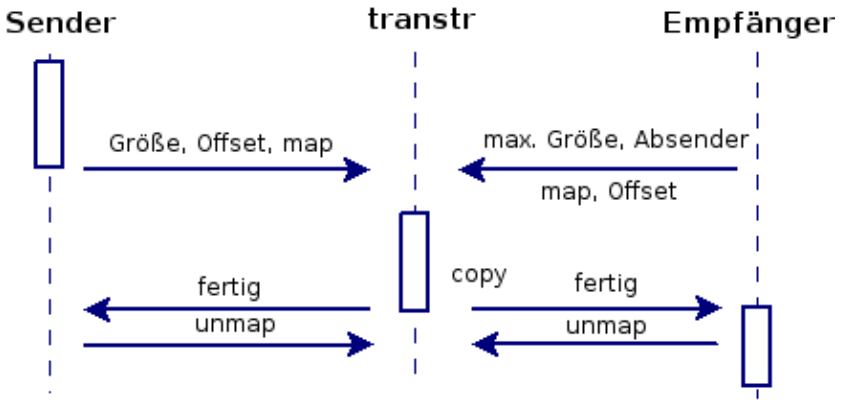

Abbildung 3.1: Zentrales Kopieren: Nachrichtenaustausch

Um seine Empfangsbereitschaft zu signalisieren, schickt der Empfänger dem Server eine der Nachrichtengröße entsprechende Anzahl an Speicherseiten. Zusätzlich wird die maximale Größe der erwarteten Nachricht und bei einem closed-wait auch der Absender übertragen. Außerdem wird ein Speicheroffset benötigt, der angibt, ab welcher Stelle auf die Speicherseiten geschrieben werden kann. Dies ist notwendig, da der für die Nachricht freie Speicherbereich nicht zwingend am Anfang einer Seite beginnt. Diese Informationen werden in einer Datenstruktur im Server abgelegt bis ein Sender dem Empfänger eine Nachricht senden will oder ein Timeout-Ereignis eintritt.

Um eine Nachricht zu senden, überträgt der Sender seinerseits die Speicherseiten, die die Nachrichten enthalten. Zusätzlich werden Angaben zum Empfänger und zur Größe der Nachricht mitgeschickt. Außerdem wird auch hier wie beim Empfänger ein Speicheroffset benötigt, damit der Server den Anfang der Nachricht finden kann.

Kann der Server einer zu sendenden Nachricht einen Empfänger zuordnen, kopiert er die Nachricht aus den Speicherseiten des Senders in die Speicherseiten des Empfängers. Tritt wärend des Kopierens ein Seitenfehler auf, muss dieser einem Client zugeordnet werden. Dazu wird in einer Datenstruktur gespeichert, in welchem Bereich des Serverspeicherbereichs wessen Seiten eingeblendet wurden. Dann wird der Seitenfehler an den Client weitergeschickt, den dieser behandeln muss. Geschieht dies nicht innerhalb einer Zeitgrenze, wird den Kommunikationspartnern eine Timeout-Nachricht geschickt und die Nachrichtenübertragung wird abgebrochen. Wurde die Nachricht erfolgreich übertragen, wird an Sender und Empfänger eine Bestätigung gesendet, die die Anzahl der übertragenen Bytes enthält.

Als Kosten für das Verbindungsprotokoll entstehen zwei lange Nachrichten, die Speicherseiten und Nachrichteninformationen enthalten. Die Speicherseiten müssen in den Serveradressraum eingeblendet werden. Zusätzlich werden zwei kurze Nachrichten zum Abschluss der Übertragung verschickt. Im Server wird immer eine Kopieroperation sowie Rechenzeit zur Verwaltung der Nachrichten und der Speicherseiten benötigt. Ein Copy-In bzw. ein Copy-Out wird nicht benötigt, da der von den Clients verwendete Nachrichtenbuffer direkt verwendet werden kann.

Eine wesentliche Optimierung wird erreicht, wenn man die Mapnachrichten einsparen kann. Der Server behält die verwendeten Seiten im eigenen Adressraum und kann sie so bei der nächsten Nachricht wieder verwenden. Dabei ist es unerheblich, ob der Client zuvor Sender oder Empfänger war, wenn ein Nachrichtenpuffer verwendet wird, der in den gleichen Speicherseiten liegt. Liegt er nur teilweise in den Seiten, müssen nur die fehlenden Seiten übertragen werden.

Die Seiten können im Server oder in den Clients verwaltet werden. Liegt die Verwaltung bei den Clients, müssen sie sich vor der ersten Nachricht beim Server registrieren und erhalten einen Speicherbereich im Adressraum des Servers. In diesen blenden sie die zur Übertragung notwendigen Speicherseiten ein. Der Server führt allerdings weiterhin die Kopieroperation durch und muss sich daher auch um auftretende Seitenfehler kümmern. Diese müssen, um behandelt zu werden, auf den Adressraum des jeweiligen Client abgebildet werden. Entweder müssen dies die Clients vornehmen oder der Server muss wissen, wo die eingeblendeten Nachrichten im Clientadressraum liegen. Wenn ein Client weitere Seiten einblenden will, muss er zuerst eine Nachricht ohne Seiten an den Server schicken, damit dieser ein Empfangsfenster auf den dem Client zugeordneten Adressraum setzen kann. Wird es notwendig, Clientbereiche aus dem Serveradressraum zu entfernen, muss der Client informiert werden. Dies erfordet nicht nur zusätzliche Nachrichten, es kann auch vorkommen, das der Clientthread nicht mehr existiert oder sogar ein anderer Thread mit der gleichen Kennung läuft. Dies kann zu Irritationen im Verbindungsprotokoll führen.

Alternativ zur clientseitigen Verwaltung kann auch der Server die vollständige Verwaltung übernehmen. Der Server speichert in einer Datenstruktur, welcher Client welche Seiten wo im Serveradressraum eingeblendet hält. Soll eine Nachricht gesendet oder

empfangen werden, überprüft der Server, ob die Seiten bereits eingeblendet sind bzw. welche Seiten noch fehlen. Die fehlenden Seiten werden dann vom Client angefordert und in das vom Server zuvor aufgesetzte Empfangsfenster eingeblendet. Seitenfehler können ebenfalls auf diese Weise einer Seite im Clientadressraum zugeordnet werden.

Die Verwaltung durch die Clients verteilt den Aufwand der Verwaltung auf die Clients. Allerdings steigt der Kommunikationsaufwand mit dem Server und die Komplexität des Protokolls stark an. Der Server überlässt den Clients Teile seines Adressraums. Ein Freiräumen bei knappwerdenden Ressourcen ist aufwändig, da die betroffenden Clients darüber informiert werden müssen. Eventuell reagiert der Client aber nicht auf Serveranfragen oder wurde in der Zwischenzeit beendet. Der Server wartet aber bis zum Ablaufen eines Zeitlimits auf eine Antwort. Der Server wird so anfälliger für Angriffe.

Übernimmt dagegen der Server die Verwaltung, steigt der Aufwand im Server. Dafür behält er volle Kontrolle über seinen Adressraum. Speicherseiten können ohne Interaktion mit einem Client verdrängt werden. Ein beendeter Thread liegt zwar auch noch weiter in der Verwaltungsstruktur, doch wird dieser einfach überschrieben, wenn die Speicherressourcen knapp werden. Ein Kommunikationsversuch, der fehlschlägt, mit einem solchen Thread ist nicht notwendig. Insgesamt dürft also die serverseitige Verwaltung die schnellere und sicherer Variante sein. Abbildung 3.2 zeigt die für die Vermittlung notwendigen Nachrichten. Die gestrichelt eingezeichneten Pfeile stellen Nachrichten dar, die nicht notwendigerweise gesendet werden müssen.

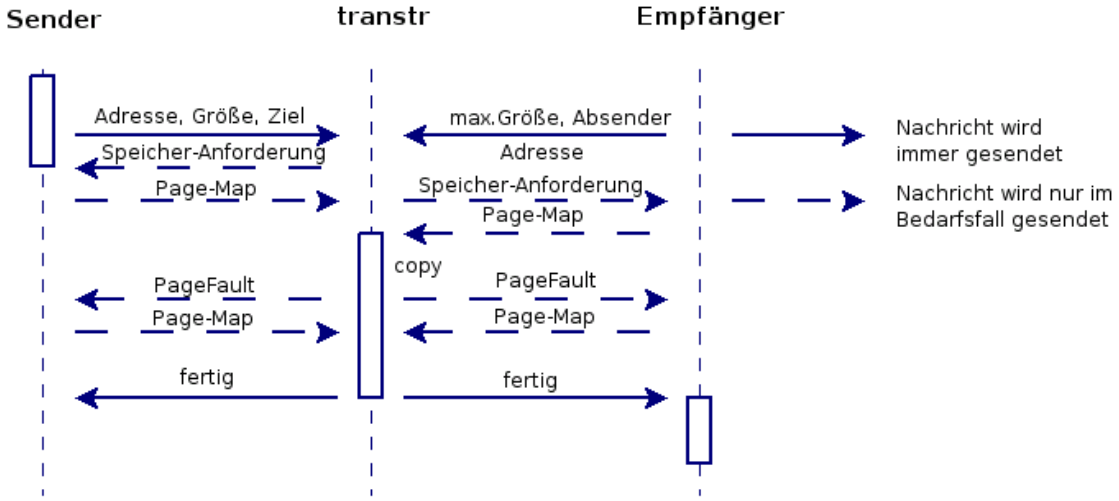

Abbildung 3.2: Zentrales Kopieren: Nachrichtenaustausch mit Seitencache

Da der Server in diesem Verfahren keine eigenen Ressourcen verteilt, können diese auch nicht mutwillig blockiert werden. Stehen keine freien Speicheradressen zum Einblenden von Nachrichten bereit, werden andere Seiten verdrängt. Behindert wird der Server, wenn einer der Clients die Speicheranfragen oder die Seitenfehler nicht korrekt beantwortet. In diesem Fall muss der Transfer nach Ablaufen einer zeitlichen Frist abgebrochen werden.

Durch die Clients gesetzte Timeouts müssen durch den Server entsprechend umgesetzt werden. Der Server kann während er auf die Beantwortung einer Speicheranfrage wartet, andere Nachrichten bearbeiten.

Ein weiterer Aspekt zur Performancesteigerung betrifft den Kopiervorgang im Server. Je nach Länge der Nachricht stellt dieser Vorgang einen relativ lange laufenden Prozess dar. Ohne Parallelisierung ist der Prozess zwar unterbrechbar, aber der Server kann keine anderen Clientanfragen beantworten. Daher sollte über Mechanismen der Parallelisierung nachgedacht werden. Die Nachrichten können in separaten Serverthreads abgearbeitet werden. Die Threads stehen im Server in einem schlafenden Zustand bereit bis sie eine Nachricht zum Kopieren erhalten. Die Threads sollten schlafen und nicht komplett beendet werden, um Zeit für die Reaktivierung zu sparen. Der zentrale Thread steht so schnell wieder bereit, um Clientanfragen bearbeiten zu können.

Löst ein Client einen Seitenfehler nicht auf, so wird die Übertragung nach Überschreiten eines Zeitlimits abgebrochen. Innerhalb dieser Zeit blockiert zwar der Empfänger, der Server darf aber nicht blockieren und muss in dieser Zeit weiter Anfragen bearbeiten.

Eine Transaktionssemantik ist einfach zu realisieren, da die Nachricht vor der Übertragung im Serveradressraum gespeichert wird. Fehler, die beim Kopieren in den Empfängeradressraum auftreten, können korrigiert werden oder die Übertragung wird vollständig abgebrochen.

Auch Multicast oder Broadcast ist möglich. Hierbei wird nicht nur ein Empfänger berücksichtigt sondern mehrere. Die Nachricht wird nur einmal an den Copyserver gesendet und wird dort mehrmals kopiert. Um gewährleisten zu können, dass alle Empfänger die Nachricht erhalten, müssen die Empfänger synchronisiert werden. Erst wenn alle empfangsbereit sind, kann die Kopieroperation gestartet werden.

3.4.2 Map Server

Bei diesem Ansatz agiert der Copy-Server als Speicherserver. Das Funktionsprinzip ist in Abbildung 3.3 dargestellt. Ein Client, der eine Nachricht verschicken will, fordert zunächst beim Server der Nachrichtengröße entsprechend Speicher an. Die Speicherseiten werden im Adressraum des Senders eingeblendet. Er kann die Nachricht dann in diesen Speicher kopieren oder sie direkt in diesem Speicher aufbauen. Der Sender signalisiert dem Server mit einer kurzen Nachricht die Fertigstellung der Nachricht. Der Server entfernt daraufhin die Speicherseiten aus dem Senderadressraum und blendet sie im Empfängeradressraum ein. Hier kann jetzt die Nachricht weiter verarbeitet bzw. heraus kopiert (copy-out) werden. Zum Schluss muss die Seite vom Empfänger wieder freigegeben werden.

Seitenfehler können im Adressraum des Empfängers und des Senders auftreten und müssen von Server behoben werden. Dies setzt den Einsatz eines Regionmappers voraus, der den Seitenfehler dem Server als Pager zuordnet. Für die Clients hat dies den Vorteil, dass die Seitenfehler von einer vertrauenswürdigen Instanz behandelt werden, allerdings wird im Server eine aufwändige Ressourcen-Verwaltung benötigt.

Es müssen freie und belegte Seiten verwaltet werden. So muss es möglich sein, belegte Seiten einem Client zuzuordnen. Eintreffende Seitenfehler enthalten nur die Adresse der Seite im Clientadressraum. Dieser muss dann auf die entsprechende Adresse im Server-

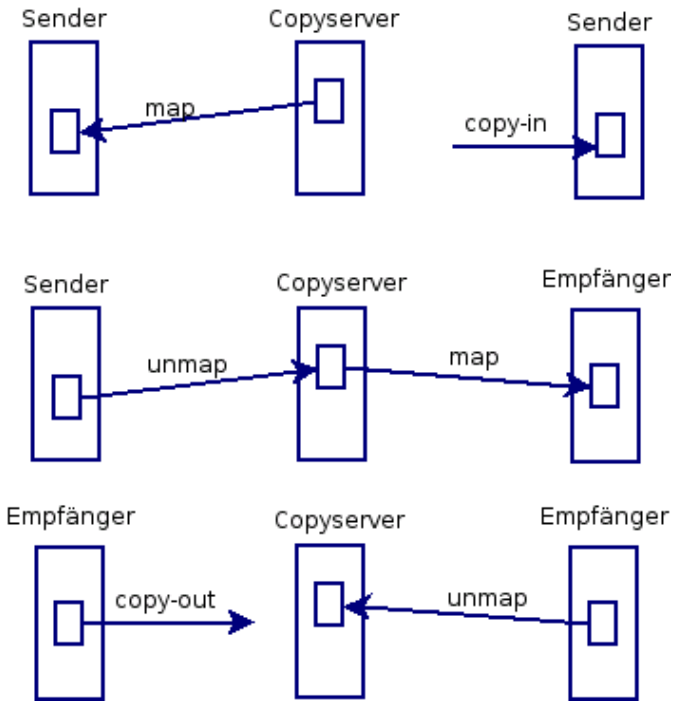

Abbildung 3.3: Zentrales Mappen: Funktionsprinzip

adressraum übersetzt werden, damit der Fehler behoben werden kann. Daher muss dem Server mitgeteilt werden, wo die Seiten im Clientadressraum eingeblendet wurden.

Desweiteren muss verhindert werden, dass ein Client den gesamten Speicher des Servers belegt und nicht wieder freigibt. Ansonsten könnten keine weiteren Nachrichten übertragen werden. Dies kann durch ein Speicherquota realisiert werden, das die maximale Speichermenge, die ein Client belegen darf, festlegt. Darüber hinaus sollten die Clients die Speicherseiten nur für eine begrenzte Zeit belegen dürfen. Diese Zeit sollte zum Aufbauen oder Auslesen der Nachrichten ausreichen. Nach Ablaufen dieser Zeit wird der Speicher zwangsweise entzogen und danach eintreffende Seitenfehler nicht mehr behoben. Dies muss dann allerdings bei der Clientprogrammierung berücksichtigt werden.

Die Übertragung kann synchron oder asynchron erfolgen. Bei einer asynchronen Übertragung wartet der Server mit der Zustellung, bis der Empfänger bereit ist, während der Sender bereits weiterarbeitet. Kann die Nachricht innerhalb eines Zeitlimits nicht zugestellt werden so wird sie verworfen. Der Sender kann so nicht sicher sein, ob die Nachricht wirklich zugestellt werden konnte. Nachfolgend wird die Signalisierung für eine synchrone Übertragung im Detail diskutiert.

Als Kosten entstehen hier zunächst die für das Protokoll benötigten Nachrichten. Als erstes muss der Empfänger dem Server seine Empfangsbereitschaft signalisieren. Dazu werden folgende Daten übertragen:

- Ein Code, der dem Server die Art der Nachricht anzeigt (1 Byte)

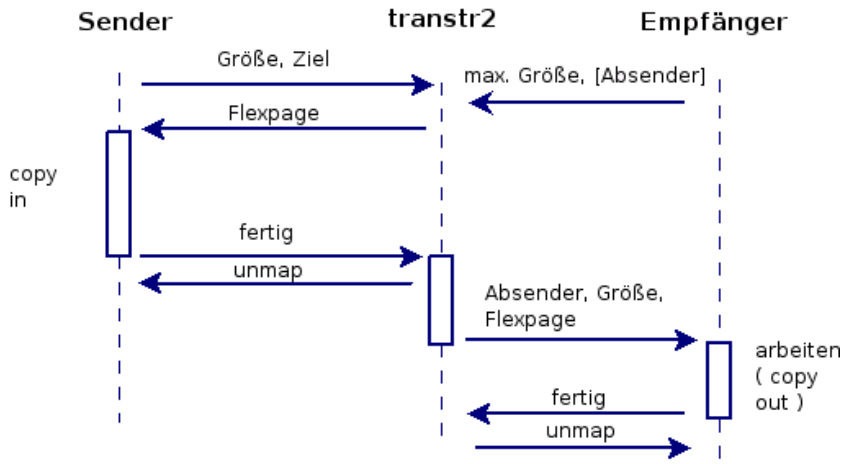

Abbildung 3.4: Zentrales Mappen: Nachrichtenaustausch

- Die maximale Größe der erwarteten Nachricht (3 Byte)
- Der erwartete Absender der Nachricht bei einem Closed-Wait (Bei L4v2 ist die ThreadID 8 Byte groß.)

Da die Absenderkennung nur bei einem Closed-Wait übertragen wird, liegt die Nachrichtengröße zwischen 4 und 12 Bytes und kann damit in L4v2 nicht immer als kurze Nachricht übertragen werden. Ein Closed-Wait erhöht damit deutlich die Kosten für das Kommunikationsprotokoll.

Um eine Nachricht zu verschicken, sendet der Sender ebenfalls entsprechende Daten zum Server:

- Ein Code, der dem Server die Art der Nachricht anzeigt (1 Byte)
- Die Größe der Nachricht (3 Byte)
- Den Empfänger der Nachricht (8 Byte)

Die Nachricht ist in jedem Fall 12 Byte groß, und damit in L4v2 für eine kurze Nachricht zu groß. In x2 passt sie allerdings in eine kurze Nachricht.

Liegt dem Server sowohl vom Sender als auch vom Empfänger eine Bereitschaftsmeldung vor, schickt er dem Sender eine zur Größe der Nachricht passende Menge an Speicherseiten und eine Übertragungskennung. Diese Kennung wird vom Sender wieder zum Server geschickt, wenn er mit dem Einschreiben der Nachricht fertig ist. Der Server kann so in der eigenen Verwaltungsstruktur die Nachricht identifizieren und einem Empfänger zuordnen. Dieser erhält jetzt die Speicherseiten und kann mit der Verarbeitung beginnen. Um die Seiten wieder freizugeben, schickt der Empfänger seinerseits dem Server eine Nachricht, die die Nachrichtenkennung enthält. So kann der Server dem Empfänger die Rechte an den Speicherseiten entziehen und sie intern als frei markieren. Zuvor muss

er aber die in den Seiten enthaltenen Daten überschreiben. Ansonsten könnte ein anderer Prozess an Informationen aus der zuvor verschickten Nachricht gelangen.

Alternativ kann der Sender den Speicher für den Sendepuffer erhalten um darin die Nachricht aufzubauen, bevor im Server ein Empfänger registriert ist. Dadurch sind die Wartezeiten für Sender und Empfänger kürzer. Es kann allerdings passieren, dass der Sender die Nachricht aufgebaut bzw. in den Sendepuffer kopiert hat und sie dann nicht zugestellt werden kann.

Insgesamt liegen die Protokollkosten so bei mindestens vier Nachrichten, wobei mindestens eine davon in L4v2 eine lange Nachricht ist. Zusätzlich werden zwei Seitenmaps benötigt. Die Kopieroperationen zählen nicht mit, da sie bei geschickter Datenverarbeitung in den Clients entfallen können. Dafür wird eine aufwändige Ressourcenverwaltung im Server benötigt, die zusätzliche Rechenzeit benötigt.

Da dem Sender der Speicher entzogen wird, kann dieser nach dem Absenden die Nachricht nicht mehr verändern. Die Nachricht wird mit den Speicherseiten vollständig an den Empfänger übertragen und die Transaktionssicherheit kann so gewährleistet werden. Der Sender kann nach dem Versenden der Nachricht nicht mehr auf die Daten zugreifen. Werden diese noch benötigt, muss vorher ein Kopie angelegt werden.

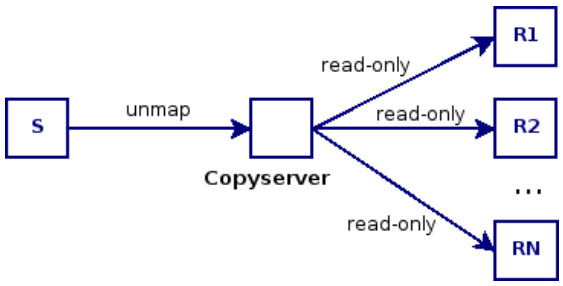

Abbildung 3.5: Zentrales Mappen: Multicast

Abbildung 3.5 zeigt wie ein Multi- bzw Broadcast realisiert werden kann. Nachdem dem Sender der Speicher entzogen wird, werden die Seiten bei allen Empfängern eingeblendet. Damit die Nachricht nicht verändert werden kann, erhalten die Empfänger nur Leserechte. Alternativ kann die Nachricht auch als copy-on-write markiert werden, so dass sich Veränderungen nicht auf die Originalnachricht im Puffer auswirken.

Eine mögliche Optimierung zielt auf das Einsparen der Mapvorgänge. Ein Empfänger kann den Empfangspuffer zum Sendepuffer der nächsten Nachricht machen. Die Freigabe und das Anfordern des Sendepufferes entfallen dabei. Dies bietet sich insbesondere dann an, wenn die gleichen Daten leicht verändert an einen andere Task weiter übertragen werden sollen. So werden Kopier- und Mapoperationen eingespart, setzt aber genaue Kentnisse bei der Programmierung der Clients voraus.

Anders sieht es aus, wenn dem Sender und dem Empfänger unterschiedliche Seiten gemappt werden und die Daten vom Server in die jeweiligen Seiten kopiert werden. Dadurch können die Mapvorgänge vollständig eingespart werden, dafür wird aber eine zusätzliche Kopieroperation benötigt. Diese Variante ist dem Entwurf, der im folgenden Kapitel diskutiert werden soll, sehr ähnlich.

Da der Copyserver den Speicherverwaltet und damit auch verantwortlicher Pager ist, kann kein Angriff durch endlose Bearbeitung von Seitenfehlern auftreten. Der dem Copyserver übergeordnete Pager muss vertrauenswürdig sein, damit die Resourcen für die Verarbeitung von Nachrichten auch verfügbar sind.

3.5 Kernkopie

In den vorangegangenen Abschnitten wurden Lösungsansätze vorgestellt, die ohne Unterstützung des Kerns auskommen. Nachfolgend sollen noch Ansätze vorgestellt werden, die zur Übertragung der Daten den Kern verwenden. Die wesentliche Aufgabe des Kerns besteht dabei darin, die Daten aus dem Empfängeradressraum in den Senderadressraum zu kopieren. Um dies durchführen zu können, muss er die entsprechenden Speicherseiten adressieren zu können. Es gibt verschiedene Wege um dies zu ermöglichen.

Wie schon in Abschnitt 2.1 kurz dargestellt, wird bei der herkömmlichen Methode zur Übertragung von langen IPC-Nachrichten der Empfangspuffer in den Kernbereich des Senderadressraums eingeblendet. Dann wird die Nachricht in diesen Puffer kopiert. Während der gesamten Übertragung werden beide Prozesse angehalten. Tritt ein Seitenfehler auf, muss dieser an den zuständigen Prozess gesendet werden. Eine aufwändige Zustandsmaschine sorgt dafür, dass nach der korrekten Behandlung des Seitenfehlers die Übertragung fortgesetzt werden kann. Da der Kernbereich im Adressraum während eines Kontextwechsels immer geleert wird, müssen für die dort eingeblendeten Seiten keine Rückverweise für das Entfernen der Seiten gepflegt werden. Dies mach das Einblenden sehr einfach.

Um Seitenfehler zu vermeiden, kann auch direkt der physische Speicher in den Kernbereich eingeblendet werden. Seitenfehler im Empfangspuffer müssen dazu vorher behoben worden sein und der physische Speicher muss in den Kernbereich passen. Dann können die Daten direkt kopiert werden.

Zum Einblenden der Speicherseiten kann auch ein komplett eigener Adressraum verwendet werden. Wie auch beim Kopier-Server müssen dabei sowohl der Sendepuffer als auch der Empfangspuffer eingeblendet werden. Diese Speicherseiten können aber für zukünftige Nachrichten eingebelendet werden. Beim Einblenden der Seiten müssen hier aber die vollständigen Strukturen gepflegt werden um die Seiten wieder entfernen zu können. Umgehen kann man dieses Problem in dem man bei einem 'unmap' Aufruf den gesamten Adressraum leert. Unter der Annahme, dass es selten zu einem unmap kommt, bleibt diese Variante effizienter.

3.6 Kern Copy-In / Copy-Out

Mit Hilfe von Rechten, die übertragbar und durch den Kern geschützt sind, lässt sich auch ein Copy-In bzw. Copy-Out realisieren, daß durch den Kern durchgeführt wird. Beim Copy-In gibt der Empfänger dem Sender das Recht, in einen bestimmten Bereich seines Speichers zu schreiben. Dieser Bereich ist nicht auf die Größe von Seiten beschränkt, sondern kann wesentlich genauer angegeben werden. Mit diesem Recht ruft der Sender einen Systemaufruf auf, woraufhin der Kern die Daten kopiert.

Bei der Kopieroperation ist nur der Senderprozess involviert. Bei einem Seitenfehler müssen also nicht wie bisher zwei Prozesse angehalten und kontrolliert werden. Der Seitenfehler wird als Fehlermeldung des Systemaufrufs zurückgegeben und der Sender entscheidet selbst über das weitere Vorgehen. So werden alle komplizierten Vorgänge außerhalb des Kerns durchgeführt. Nur die Kopieroperation wird im Kern durchgeführt.

3.7 Auswahl

Das zentrale Kopieren zeichnet sich gegenüber dem zentralen Mappen durch einen niedrigeren Verwaltungsaufwand aus, da keine Ressourcen verwaltet werden müssen. Lediglich Speicheradressen und von den Clients eingeblendete Seiten müssen verwaltet werden. Durch die geringere Komplexität ist dieser Server auch weniger anfällig für Denial-of-Service-Angriffe, insbesondere da keine Ressource knapp werden kann. Der für eine Task zur Verfügung gestellte Adressbereich kann sehr einfach beschränkt werden. Dafür wird für jede Nachricht eine Kopieroperation im Server notwendig, die mit zusätzlichem Aufwand parallelisiert werden muss. Die Kopieroperationen werden, wenn sie überhaupt notwendig sind, beim zentralen Mappen von den Clients durchgeführt.

Sehr hohe Kosten entstehen auch beim Mappen von Speicherseiten, was bei beiden Methoden notwendig ist. Beim zentralen Kopieren ist das Mappen aber mit recht wenig Aufwand sehr häufig einsparbar. So fällt dies weg, wenn ein Client häufig lange Nachrichten sendet oder empfängt. Beim zentralen Mappen ist dies nur einsparbar, wenn ein Client eine Nachricht erst empfängt und dann die gleichen Speicherseiten zum Senden einer Nachricht verwendet. Diese Optimierung muss aber im Client implementiert sein, um sie nutzbar zu machen. Eine transparente Implementierung durch den Server ist hier nicht möglich.

Für eine einfache Geschwindigkeitsmessung sollen beide Lösungen implementiert werden. Dabei sollen zunächst Sicherheitsmechnismen vernachlässigt werden, die bei der normalen Übertragung keinen Einfluss auf die Übertragungsgeschwindigkeit haben. Die Messungen sollen dann zeigen, welches Verfahren tatsächlich schneller ist. Da bei der PingPong-Messung immer ein Thread einem anderen, immer gleich bleibenden Thread, eine Nachricht schickt, wird das zentrale Kopieren durch das komplette Einsparen der Mapnachrichten einen Vorteil haben. Allerdings muss sich zeigen, ob das zentrale Mappen dies durch Einsparen des Kopierens der Nachricht wieder ausgleichen kann.

Um das System unter Last testen zu können, soll es auch eine Messung unter realitätsnahen Bedingungen geben. Da das zentrale Kopieren im Aufbau einfacher ist, soll diese Lösung soweit ausgebaut werden. Dazu soll der neue Übertragungsweg für die Anwendungen transparent in die bestehende IPC-Implementierung integriert werden. Alle Nachrichten, die eine entsprechende Länge haben, werden dann über den Server übertragen.

4 Implementierung

In diesem Kapitel wird die Implementierung der in 3.4.1 und in 3.4.2 vorgestellten Entwürfe kurz beschrieben. Schwerpunkt der Implementierung war ein für Performancemessung geeignetes System. Daher wurden Sicherheitsmechnismen vernachlässigt, die auf die Performance keinen Einfluss haben.

Die beiden Ansätze realisieren die Kommunikation über einen zentralen Server, der als Vermittler zwischen den Kommunikationspartnern steht. Die beiden Ansätze unterscheiden sich in der Art, wie die Nachricht zwischen Client und Server ausgetauscht wird.

4.1 Zentrales Mappen

Kernaufgabe des Servers ist die Verwaltung der Speicherseiten, die für den Nachrichtentransport verwendet werden. Die Ressourcenverwaltung muss den Clients die richtige Anzahl an Seiten zuweisen und darüber wachen, dass kein Client zu viel Speicher für zu lange Zeit blockiert. Außerdem müssen auftretende Pagefaults einem Client zugeordnet und die Seitenadresse auf den Adressraum des Clients abgebildet werden.

In der vorliegenden Implementierung werden Nachrichten vom Sender auch entgegengenommen, wenn noch kein bereiter Empfänger vorliegt (siehe Kapitel 3.4.2).

Die Nachrichten werden im Server zusammen mit den Speicherseiten in einer Datenstruktur verwaltet. Jeder Nachricht wird dabei ein Empfänger und ein Sender zugeordnet. Sobald der Nachrichtenaustausch beginnt, werden der Nachricht die verwendeten Speicherseiten zugeordnet. Zu jeder im Server zum Nachrichtenaustausch verfügbaren Speicherseite existiert ein Eintrag in der Datenstruktur, in dem die Adresse der Seite und ihre Belegung gespeichert wird.

Eine im Server registrierte Nachricht kann sich in einem von vier verschiedenen Zuständen befinden:*wait*, *send*, *pending* und *recv*. Anhand des Zustands erkennt der Server, welcher Client gerade den Nachrichtenspeicher eingeblendet hat oder auf welche Clientaktion gerade gewartet wird. Der Aufbau der Verwaltungsstruktur verändert sich mit jedem Zustand. Nachfolgend sind die Zustände ausführlich erläutert.

wait Der Empfänger hat sich beim Server als empfangsbereit gemeldet und es lag keine versandbereite (pending) Nachricht vor. Im Server wird die Nachrichtenstruktur angelegt, in die der Empfänger und die maximale Nachrichtengröße gespeichert wird. Zusätzlich wird hier im Falle eines closed-wait der Sender gespeichert. Speicher wird der Nachricht noch nicht zugewiesen, da dieser noch nicht benötigt wird.

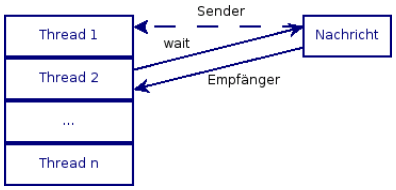

Abbildung 4.1: Datenstruktur im Zustand 'wait'

send Ein Client möchte eine Nachricht senden und hat vom Server dazu Speicherseiten erhalten. In der Verwaltungsstruktur wird der Sender, der Empfänger und die Nachrichtengröße gespeichert. Außerdem wurden der Nachricht eine oder mehrere Speicherseiten zugeordnet. Der Sender kann jetzt die Nachricht in diesem Speicher zum Versand vorbereiten. Wird dieser Zustand eingeleitet, obwohl noch kein bereiter Empfänger im Server registriert ist, so wird die Nachricht zusätzlich in der Pending-List eingetragen. Ansonsten würde die Nachricht nicht berücksichtigt werden, wenn sich der Empfänger registriert.

pending Der Sender hat die Nachricht fertigstellt und zum Versand freigegeben. Allerdings steht noch kein passender Empfänger bereit. Die Nachricht wird als versandbereit markiert und in einer extra Liste, der Pending-List, gespeichert. Die Speicherseiten werden aus dem Adressraum des Senders entfernt.

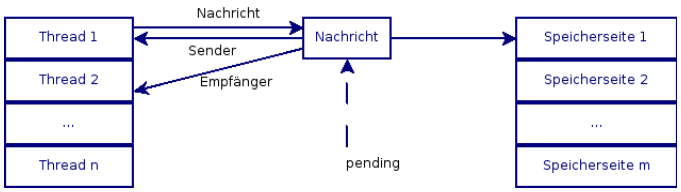

Abbildung 4.2: Datenstruktur im Zustand 'send' und 'pending'

recv Der Speicher wurde in den Empfängeradressraum eingeblendet und der Sender wurde über die Zustellung der Nachricht informiert. Der Empfänger kann jetzt die empfangene Nachricht weiter verarbeiten und muss zum Schluss mit einer Nachricht an den Server den Speicher wieder freigeben.

Gibt der Empfänger den Speicher frei, wird die zur Nachricht gehörende Datenstruktur gelöscht. Die verwendeten Speicherseiten werden wieder als „Frei“ markiert. Die enthaltenen Daten werden gelöscht um anderen Threads das Auslesen der Nachricht nicht zu ermöglichen.

Die Implementierung ist nur zur Messung der Performance vorgesehen. Daher wurden keine weitreichenden Sicherheitsmechanismen implementiert. So werden keine Timeout-Bedingungen überwacht und Timeout-Ereignisse ausgelöst. Außerdem gibt es keine

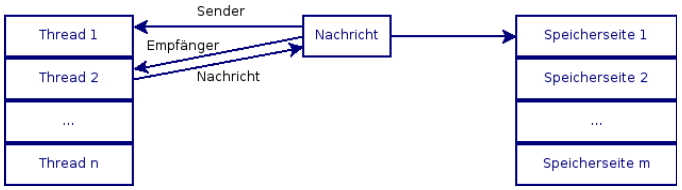

Abbildung 4.3: Datenstruktur im Zustand 'recv'

Begrenzung für der Speichermenge, die ein Client belegen kann. Eine Attacke auf den Server kann so den gesamten verfügbaren Speicher des Servers blockieren und damit andere Tasks behindern.

Erwartungsgemäß wird ein sehr hoher Aufwand für die Verwaltung der Speicherressourcen benötigt. Außerdem müssen für jede Nachricht Speicherseiten versendet werden. Insgesamt ergibt sich so ein hoher Ressourcenaufwand und eine Anfälligkeit für Angriffe. Wie sich der Ressourcenaufwand auf die Übertragungsgeschwindigkeit auswirkt, wird sich bei der Performancemessung zeigen.

4.2 Zentrales Kopieren

Bei diesem Ansatz wird die Nachricht zwischen Client und Server über einen geteilten Speicher ausgetauscht. Der Server kopiert die Nachricht aus dem Speicher des Senders in den Speicher des Empfängers. Zur Verbesserung der Performance bleibt der geteilte Speicher im Adressraum des Servers eingeblendet, um ihn für weitere Nachrichten weiter nutzen zu können.

Der Server kann von den Clients drei unterschiedliche Anfragen empfangen:

- Der Sender möchte eine Nachricht verschicken.
- Der Empfänger meldet sich emfangsbereit.
- Der Empfänger widerruft seine Empfangsbereitschaft.

Unterschieden werden diese Nachrichten über das erste Datenwort. In diesem überträgt der Sender, wenn er eine Nachricht senden will, die Adresse seines Sendepuffers. Liegt dieser Wert außerhalb des zulässigen Adressbereichs, stammt die Nachricht vom Empfänger. Bei der vorliegenden Implementierung für den Fiasco-Kern sind dies alle Adressen, die größer als 0xFF000000 sind. Dieser Bereich ist für den Kern reserviert. In dieser Zahl integriert der Sender einen Code, der dem Server anzeigt, ob der Client zum Empfang bereit ist, oder eine vorherige Bereitschaftsmeldung widerrufen will.

Die Integration wird durch einfache Addition realisiert. Auf den Basiswert 0xFF000000 wird zur Bereitschaftsanmeldung die Nachrichtengröße addiert. Zum Widerrufen einer solchen Meldung wird als Code einfach eine Zwei addiert. Da über den Server keine so kurzen Nachrichten geschickt werden, kann die Zwei nicht mit einer Größenangabe verwechselt werden. Bei einer Bereitschaftsmeldung wird zusätzlich im zweiten Datenwort die Adresse des Empfangspuffers übertragen.

Anhand der empfangenen Adresse und Nachrichtengröße ermittelt der Server die Menge der Speicherseiten, die noch vom Client angefordert werden müssen. Bereits im Serveradressraum eingeblendete Seiten müssen nicht erneut angefordert werden. Um Seiten vom Client anzufordern, wird diesem eine kurze Nachricht geschickt. Da dem Client unterschiedliche Anfragen gesendet werden können, enthält die Nachricht eine Codezahl. Zusätzlich wird die Adresse der ersten angeforderten Seite und die Menge der Seiten übertragen. Neben Speicheranforderungen werden so auch Seitenfehler und Übertragungsbestätigungen an die Clients geschickt.

Im Server werden die Threads und ihrer Speicherbereiche in einer mehrstufigen Verwaltungsstruktur erfasst. Abbildung 4.4 zeigt schematisch den Aufbau dieser Struktur. Die erste Datenstruktur speichert die einzelnen Threads mit ihren Thread-IDs, über die die Threads identifiziert werden. Jeder der hier erfassten Threads hat eine Liste mit Speicherbereichen, die im Serveradressraum eingeblendet sind. Gespeichert wird die Adresse der ersten Seite im Clientadressraum und die Anzahl der eingeblendeten Seiten. Die Adresse im Serveradressraum, in der Abbildung 4.4 auch virtuelle Adresse genannt, wird in einer eigenen Struktur gespeichert. Dieser Aufbau erleichtert das Auffinden bei einem Seitenfehler. Hierbei wird in der Liste nach der Adresse des Seitenfehlers gesucht, die dann auf die echte Adresse und auf den zuständigen Client verweist. An diesen wird dann der Seitenfehler mit der Adresse im Clientadressraum weitergeleitet.

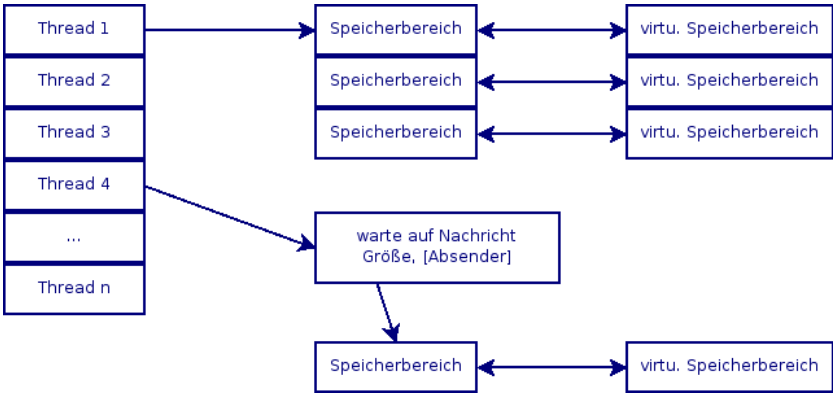

Abbildung 4.4: Zentrales Kopieren: Speicherverwaltung im Server

Die Datenstruktur enthält aber auch Informationen über erwartete Nachrichten eines Empfängers. Dazu verweist der Thread auf Angaben über die Nachricht wie Größe und, bei einem closed-wait, den Absender. Für die spätere Zustellung der Nachricht wird zusätzlich auf die für den Empfang zu verwendenden Speicherbereich mit seiner virtuellen Adresse verwiesen. Trifft eine für diesen Thread passende Nachricht beim Server ein, wird die Nachricht direkt zugestellt und der Warteeintrag wird entfernt.

Will ein Sender eine Nachricht an einen Empfänger übertragen, der noch nicht bereit ist, wird diese Nachricht in einer speraten Liste gespeichert. Trifft dann die Bereitschaftsmeldung des Empfängers ein, kann die Nachricht sofort zugestellt werden. Die Verweildauer einer Nachricht in einem der beiden wartenden Zustände soll durch Timeout-Regelungen begrenzt werden. Nach Ablaufen einer vorgegebenen Zeitspanne wird die

Verbindung abgebrochen, der Client bekommt eine entsprechende Meldung. Dies ist in der vorliegenden Implementierung noch nicht realisiert, da dies für eine Performancebeurteilung nicht relevant ist.

Der Lösungsansatz über einen Server, der die Nachricht zwischen den Speicherseiten der Clients kopiert, zeichnet sich durch einen geringen Aufwand für die Verwaltung und ein recht einfaches Kommunikationsprotokoll aus. Dadurch entsteht auch eine recht hohe Robustheit gegen Angriffe.

5 Auswertung

5.1 Geschwindigkeitsmessung in PingPong

Die Übertragungsgeschwindigkeit wird mit dem PingPong Server gemessen. Dieser erzeugt zwei Prozesse und schickt von einem zum anderen Nachrichten unterschiedlicher Länge. Zur Minimierung von Messfehlern wird die Zeit zur Übertragung von mehreren Nachrichten gemessen und der Mittelwert bestimmt.

Gemessen wird die Geschwindigkeit bei der Übertragung über einen Copy-Server wie in 3.4.1 und 4.2 beschrieben und über eine Map-Server wie in 3.4.2 und 4.1 vorgestellt. Die Ergebnisse werden mit dem kernbasierten long-IPC Mechanismus verglichen.

Für die Messungen wurde in Computer mit einem Pentium 4 Prozessor mit 2793 MHz. Dieser hat 16 Kilobyte Level eins Cache und 1 MB Level zwei Cache.

5.1.1 Copy-Server

Bei der Übertragung über den Copy-Server werden Speicherseiten aus dem Adressraum des Senders und des Empfängers in den Adressraum des Copy-Servers eingeblendet. Der Copy-Server kopiert dann die Nachricht. Wie in 3.4.1 dargestellt werden für das Protokoll vier IPC-Nachrichten benötigt, von denen in der Implementierung in l4v2 mindestens eine eine lange IPC-Nachricht ist. Zu den so entstandenen Kosten kommen noch Kosten für Kontextwechsel und für den Verwaltungsaufwand im Server.

Da bei diesem Messverfahren viele Nachrichten von einem Sender zu einem Empfänger übertragen werden, wird nur die reine Übertragungszeit erfasst. Die Speicherseiten werden nur einmal an den Server übertragen und stehen dann für die gesamte Messzeit zur Verfügung. Es treten also auch keine Seitenfehler auf. Tabelle 5.1 zeigt die Ergebnisse dieser Messungen und zum direkten Vergleich die Zeiten für die kernbasierte long-IPC.

Nachrichtenlänge	Geschw. Transtr	Geschw. Kern
8 Worte	24573 Takte	5311 Takte
16 Worte	24640 Takte	5325 Takte
64 Worte	24764 Takte	5467 Takte
256 Worte	25349 Takte	5994 Takte
1024 Worte	27399 Takte	8188 Takte
2048 Worte	30382 Takte	11381 Takte

Tabelle 5.1: PingPong-Messung der Übertragung über den Copy-Server

Um die entsandenen Kosten genauer lokalisieren zu können, ist es notwendig, diese in den einzelnen Schritten der Übertragung zu messen. In Abbildung 5.1 sind diese Einzelschritte mit Nummern markiert. Jeweils vor und hinter jedem Messabschnitt wurde der Stand des CPU-Timers ausgelesen und die Differenz errechnet. Lagen die Messpunkte in unterschiedlichen Prozessen, zum Beispiel wenn eine Protokollnachricht ausgmessen werden soll, erfolgt die Errechnung außerhalb des Programms. Dazu werden die ausgelesenen Werte nach Verlassen des Messstrecke ausgegeben. Jede dieser Teilmessungen wurde mehrmals wiederholt und der Mittelwert errechnet.

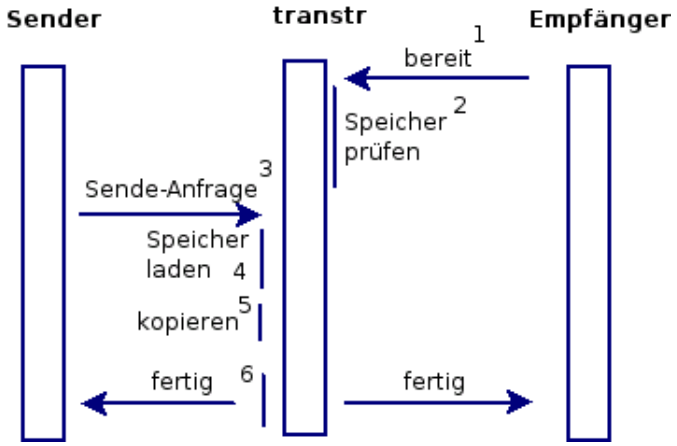

Abbildung 5.1: Messpunkte bei der Übertragung über den Copy-Server

Legende zu Abbildung 5.1

1: WAIT-Anfrage Der Empfänger signalisiert seine Empfangsbereitschaft und überträgt die maximale Nachrichtenlänge und die Adresse des Empfangspuffers. Der erste Messpunkt liegt im Empfänger kurz vor dem Absenden der Nachricht. Der zweite Messpunkt liegt im Transtr-Server nach dem Eintreffen der Nachricht.

2: Empf.-Speicher prüfen Der Transtr-Server prüft, ob die Speicherseiten mit dem Empfangspuffer bereits eingeblendet sind. Im Messzenario ist dies stets der Fall. Zur Prüfung werden aus der Datenstruktur anhand der ThreadID die bereits eingeblendeten Seiten eingelesen.

3: SEND-Anfrage Der Sender signalisiert seinen Sendewunsch und überträgt die Nachrichtenlänge, die Sendepufferadresse und die ThreadID des Empfängers. Wie auch schon bei der WAIT-Anfrage wird jeweils im Sender und im Transtr-Server gemessen.

4: Speicher laden Es wird geprüft ob sich bereits ein passender Empfänger im Transtr-Server angemeldet hat. Dann wird geprüft, ob der Sendepuffer vollständig im Serveradressraum eingeblendet ist. Beide Prüfungen führen in der Messumgebung zu einem positiven Ergebnis.

5: Kopieren Hier wird die Zeit gemessen, die benötigt wird, um die Nachricht aus dem Sendepuffer in den Empfangspuffer zu kopieren.

6: Abschluss-Meldung Die Messpunkte umfassen die Sendebefehle für das Senden der Abschluss-Meldungen an den Sender und den Empfänger durch den Transtr-Server.

Nachrichtenlänge (Worte)	**8**	**16**	**64**	**256**	**1024**	**2048**
1: WAIT-Anfrage	2751	2644	2653	2662	2652	2669
2: Empf.-Speicher prüfen	140	140	140	140	140	140
3: SEND-Anfrage	5877	5601	5592	5589	5572	5562
4: Speicher laden	624	624	624	624	624	624
5: kopieren	252	278	420	983	3158	6099
6: Abschluss-Meldung	10641	10663	10631	10637	10649	10687

Tabelle 5.2: Messergebnisse der Teilmessungen der Übertragung mit dem Copy-Server

Wie zu erwarten hängt nur die Kopieroperation von der Länge der Nachricht ab, alle anderen Werte bleiben in etwa gleich. Die Kosten für 2 und 4 hängen von den Kosten für den Zugriff auf die Datenstruktur im Transtr-Server ab. Insbesondere das Auffinden des richtigen Threads in dieser Datenstruktur hängt von der Menge der Threads ab, die den Transtr-Server benutzen. Bei der vorliegenden Realisierung durch verzeigerte Listen steigt der Aufwand für den Zugriff linear zur Menge der Threads. Eine optimierte Implementierung kann hier Kosten sparen. In der Messumgebung bleibt die Anzahl allerdings konstant und damit auch die Kosten für den Zugriff.

Die hohen Kosten für das Versenden der Abschluss-Meldungen entstehen durch die benötigten Kontext-Wechsel. Zunächst wird die Nachricht an den Sender geschickt. Wenn dieser Aufruf zurückkehrt und damit in den Kontext des Transtr-Servers zurück gewechselt wurde, wird eine Nachricht an den Empfänger geschickt. Der zweite Messpunkt wird erst erreicht, wenn ein weiteres Mal in den Kontext des Transtr-Server gewechselt wird. Da die Messimplemtierung des Transtr-Servers keine Parallelität vor sieht, steht der Server auch erst jetzt für die nächste Nachricht bereit. Parallelität hätte es aber auch erschwert, die korrekten Kosten für die Übertragung einer Nachricht zu bestimmen.

5.1.2 Map-Server

Bei der Übertragung mit Hilfe des Map-Servers, hier auch Transtr2-Server genannt, stellt der Server den Speicher für die Nachricht zur Verfügung. Er wird zunächst im Adressraum des Senders eingebelendet. Dieser kopiert die Nachricht in diesen Puffer und bekommt die Rechte dafür wieder entzogen. Dann wird der Speicher im Empfängeradressraum eingeblendet, bis dieser die Verarbeitung der Nachricht abgeschlossen hat. Der Aufbau und die Implementierung ist in Abschnitt 3.4.2 und 4.1 detailliert beschrieben.

Der Aufwand für Protokollnachrichten entspricht dem aus dem Transtr-Server. Allerdings kommen bei dem Transtr2-Server die Kosten für das Mappen von Speicherseiten

hinzu. Zusätzlich ist die Verwaltung im Server aufwändiger, so dass auch hier mit höheren Kosten zu rechnen ist.

Der Messaufbau entspricht dem, der im vorangegangenen Abschnitt verwendet wurde. Tabelle 5.3 zeigt die Ergebnisse der Messung der Übertragung von Nachrichten mit Ping-Pong. Zur Lokalisierung der Kosten wurden auch bei diesem Ansatz an verschiedenen Punkten Messungen vorgenommen. Abbildung 5.2 zeigt die Verteilung der Messstellen und die Tabelle 5.4 die Ergebnisse der Messungen und zum direkten Vergleich die Zeiten für die kernbasierte long-IPC.

Nachrichtenlänge	Geschw. Transtr2	Geschw. Kern
8 Worte	80124 Takte	5311 Takte
16 Worte	80775 Takte	5325 Takte
64 Worte	81152 Takte	5467 Takte
256 Worte	81790 Takte	5994 Takte
1024 Worte	86789 Takte	8188 Takte
2048 Worte	100766 Takte	11381 Takte

Tabelle 5.3: PingPong-Messung der Übertragung über den Map-Server

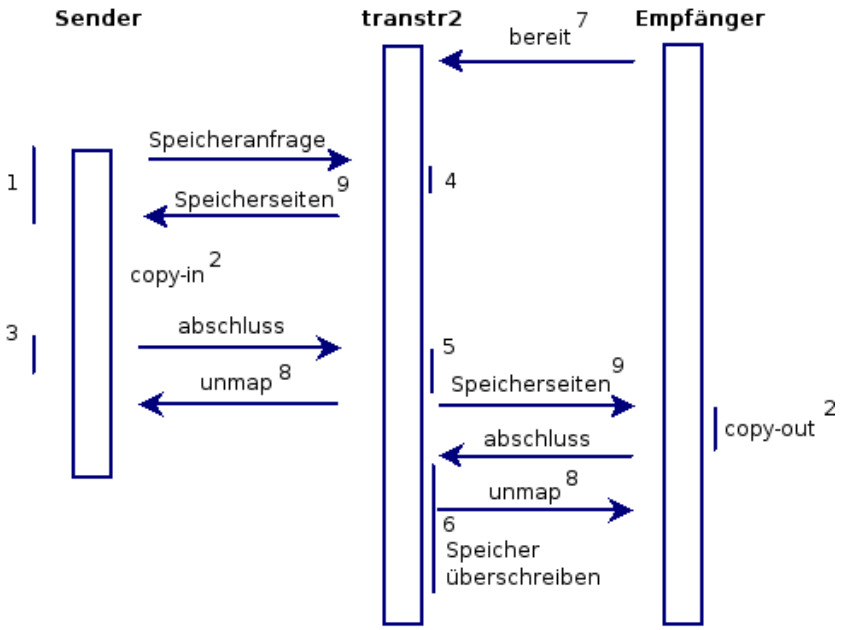

Abbildung 5.2: Messpunkte bei der Übertragung über den Map-Server

Legende zur Abbildung 5.2

1: Speicheranfrage Der Sender signalisiert dem Transtr2-Server, dass er eine Nachricht übertragen möchte. Dabei wird die Größe und die ThreadID des Empfängers übertragen. Als Antwort schickt der Server Flexpages mit den benötigten Speicherseiten. Die Messung umfasst den Zeitraum vom Absenden der Anfrage bis zum Eintreffen der Antwort im Sender. Sie überlappt mit den Messungen 4 und 9.

2: copy-in / -out Diese Messung bestimmt die benötigte Zeit für das Kopieren der Nachricht in den Puffer bzw. aus dem Puffer heraus.

3: Abschluss Hier werden die Kosten erfasst, die der Sender und der Empfänger jeweils benötigen, um dem Server zu signalisieren, dass die Bearbeitung der Nachricht in ihren Adressräumen abgeschlossen ist.

4: Speicheranfrage bearbeiten Diese Messung umfasst die Zeit vom Eintreffen der Anfrage des Senders im Server bis zum Absenden der Antwort.

5: Nachricht im Server In der hier gemessenen Zeit überprüft der Server, ob ein Empfänger registriert ist. In der Messumgebung ist dies stets der Fall.

6: Abschluss im Server In diesem Schritt wird durch Aufräumarbeiten im Server die Übertragung abgeschlossen. Dem Empfänger werden die Rechte an den Speicherseiten entzogen und anschließend der Inhalt der Speicherseiten überschrieben. Der Messzeitraum beginnt mit dem Eintreffen der Abschlussmeldung vom Empfänger.

7: WAIT-Anfrage Der Empfänger signalisiert seine Empfangsbereitschaft und überträgt die maximale Nachrichtenlänge.

8: Unmap Die Kosten für das Entziehen der Rechte an den Speicherseiten wurden bestimmt, indem die Zeit für den entsprechenden Aufruf im Transtr2-Server gemessen wurde.

9: Speicherseiten mappen Die Kosten, die durch das Einblenden von Speicherseiten im Sender- bzw. im Empfängeradressraum entstehen.

Wie erwartet sind die Kosten deutlich höher als beim Transtr-Server. Insbesondere das Mappen der Speicherseiten verursacht extrem hohe Kosten, die von der Länge der Nachricht abhängen. Die Verwaltungskosten hängen vor allem von der Menge der Threads ab, die den Server verwenden. In der Messanordnung war diese Zahl allerdings konstant.

Nachrichtenlänge (Worte)	**8**	**16**	**64**	**256**	**1024**	**2048**
1: Speicheranfrage	33374	33367	33373	33398	33484	37217
2: copy-in / -out	259	308	630	1100	3244	6141
3: Abschluss	2430	2230	2324	2102	2098	2240
4: Speicheranfr. bearb.	441	441	441	441	441	441
5: Nachricht im Server	446	446	446	446	446	446
6: Abschluss im Server	5462	5463	5464	5465	5463	5464
7: WAIT-Anfrage	2320	2318	2320	2322	2324	2325
8: Unmap	3877	3877	3877	3877	3878	3878
9: Speicherseiten mappen	11039	11039	11039	11039	11039	11039

Tabelle 5.4: Messergebnisse der Teilmessungen der Übertragung mit dem Map-Server

5.2 Ergebnis

Durch die Messungen wurde festgestellt, dass beide Lösungsansätze deutlich langsamer als das kernbasierte Verfahren sind. Verursacht wird dies vor allem durch den Aufwand für Protokollnachrichten. Alle Signale müssen über IPC-Nachrichten ausgetauscht werden. Eine Verbesserung ist schon erreichbar, wenn auch die mit 16 Byte längsten Protokollnachrichten als kurze Nachrichten versendet werden können. Dies ist durch die Einsatz von L4x2 möglich. Dies würde bei der hier verwendeten Versuchsanordnung eine Einsparung von etwa 3000 Takten bringen. Bei eine Closed-Wait würden sogar 6000 Takte eingespart.

Bei der Lösung über einen Map-Server kommen noch die hohen Kosten für das Mappen der Speicherseiten hinzu. Pro Nachricht müssen die Speicherseiten zweimal gemappt werden. Eingespart kann unter idealen Bedingungen nur einer dieser Mapvorgänge. So entstehen deutlich höhere Kosten.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Die Inter-Prozess-Kommunikation (IPC) stellt einen sehr performancekritischen Teil eines mikrokernbasierten Betriebssystems dar. In dieser Arbeit wurden alternative Mechanismen zur Übertragung von längeren IPC-Nachrichten vorgestellt. Dazu wurde zunächst in Abschnitt 1.3 die Länge von Nachrichten, wie sie in einem konkreten System auftreten untersucht.

Die Untersuchung ergab, dass die Länge der Nachrichten stark von der Auslastung und der Verwendung des Systems abhängt. Dennoch kann man davon ausgehen, dass die meisten Nachrichten kürzer als 30 Byte sind. Wie in Abschnitt 2.2 dargestellt, verschiebt die neuere L4x2-Schnittstelle diese Grenze auf 64 Byte. Bei L4v2 lag sie noch bei 8 Byte, wodurch ein Großteil aller Nachrichten als lange Nachrichten behandelt wurde.

Die für die Übertragung von Nachrichten benötigten Anforderungen zum Beispiel an die Sicherheit wurden in Abschnitt 3.1 aufgestellt. Bei der weiteren Entwicklung wurde insbesondere die Sicherheit unter Berücksichtigung der verschiedenen Vertrauensbeziehungen (Abschnitt 3.1.5) beachtet. Diese Vertrauensbeziehungen erlauben es, gewisse Sicherheitsaspekte außer Acht zu lassen und somit die Performance zu verbessern.

Die vorgestellten Mechanismen wurden unterteilt in direkte (Abschnitt 3.3) und indirekte (Abschnitt 3.4) Kommunikation. Bei der direkten Kommunikation schickt der Sender die Nachricht direkt an den Empfänger. Dies kann über Sequenzen kurzer Nachrichten oder über geteilten Speicher, der auch als Ringpuffer fungieren kann, geschehen.

Die indirekte Kommuniktion (Abschnitt 3.4) verwendet einen dritten Prozess als vermittelnden Server. Dieser ist zwischen Empfänger und Sender geschaltet und führt die Datenübertragung durch. Zwischen den Kommunikationspartnern und dem Server erfolgt die Datenübertragung als direkte Kommunikation, allerdings mit besonderen Vertrauensvorbedingungen. Hierfür wurden zwei verschiedene Ansätze diskutiert. Bei der Lösung mit einem Copy-Server (Abschnitt 3.4.1) wird der Speicher der Clients verwendet und der Server kopiert die Nachricht. Bei dem Map-Server (Abschnitt 3.4.2) tritt der Server als Speichermanager auf und reicht die Seiten mit der Nachricht vom Sender zum Empfänger. Beide Lösungen wurden prototypisch implementiert (Erläuterung in Kapitel 4) und ihre Performance gemessen. Die Ergebnisse der Messugen aus Kapitel 5 sind im nachfolgenden Abschnitt kurz zusammengefasst.

Zwei kernbasierte Varianten (Abschnitt 3.5 und 3.6) wurden vorgestellt. Da diese Arbeit ihren Schwerpunkt auf nicht-kernbasierte Methoden hat, wurden diese Ansätze nicht weiter vertieft.

6.2 Ergebnis

Die Messungen aus Kapitel 5 zeigen deutlich, dass die reinen nicht kern-basierten Lösungen deutlich langsamer sind als die herkömmliche kernbasierte Lösung. Der Map-Server benötigt wegen des teueren Mappens der Speicherseiten sogar noch mehr Zeit. Ansonsten entstehen die hohen Kosten vor allem durch die für das Übertragungsprotokoll benötigten Nachrichten und die damit verbundenen Kontextwechsel. Ein solcher Kontextwechsel ist deutlich teurer als ein Kerneintritt.

Es ist also möglich, den Mechnismus zur Übertragung von langen IPC-Nachrichten aus dem Kern auszulagern und ihn somit zu vereinfachen. Dadurch kann auch dieser performancekritische Teil des Betriebssystems für verschiedene Sicherheitsstufen unterschiedlich realisiert werden. Nicht vertrauenswürdige Programme können einen anderen IPC-Server zugewiesen bekommen als funktions- oder sicherheitskritische Anwendungen. Diesen Vorzug bezahlt man aber mit höheren Übertragungskosten.

6.3 Zukünftige Arbeiten

Die angefertigte Implementierung eignet sich nur für Messungen und zur Beurteilung des Lösungsansatzes. Für einen produktiven Einsatz mangelt es noch an einer Reihe von Sicherheitsmechanismen, die auch im Fehlerfall ein reibungsloses Funktionieren gewährleisten sollen. Die fehlenden Mechanismen im Detail wurden im Kapitel 4 aufgeführt.

Auch bietet die Implentierung noch Spielraum für Optimierungen. Insbesondere die in den Servern verwendete Datenstrukturen sollten für einen Hochlast-Einsatz optimiert werden, damit der Server bei vielen Clients besser skaliert.

Ob die vorgestellten Ansätze für den produktiven Einsatz geeignet sind, hängt vom Einsatzschwerpunkt des Systems ab. Die Mechanismen bieten durch ihre Auslagerung aus dem Kernraum einige Sicherheitsvorteile. Dafür ist die Übertragung aber teurer. So muss im Einzelfall geprüft werden, ob die höhere Sicherheit den Verlust der Performance rechtfertigt.

Anhang A

Ausgewertete Funktionsschnittstellen

Nachfolgend sind in einer Tabelle alle Funktionsschnittstellen dargestellt, die für die Untersuchung in Abschnitt 1.3 verwendet wurden.

Name	**Interface**	**Funktion**
l4vfs	basic_io	open
l4vfs	basic_io	creat
l4vfs	basic_io	unlink
l4vfs	basic_io	rmdir
l4vfs	basic_io	lseek
l4vfs	basic_io	fsync
l4vfs	basic_io	getdents
l4vfs	basic_io	stat
l4vfs	basic_io	access
l4vfs	basic_name_server	resolve
l4vfs	basic_name_server	rev_resolve
l4vfs	basic_name_server	thread_for_volume
if_l4dm	bootmod	open
if_l4dm	bootmod	size
l4vfs	common_io_notify	read_notify
l4vfs	common_io_notify	write_notify
l4vfs	common_io	read
l4vfs	common_io	write
l4vfs	common_io	close
l4vfs	common_io	ioctl
l4vfs	common_io	fcntl
l4vfs	connection	init_connection
l4vfs	connection	close_connection
l4vfs	container_io	mkdir
if_l4dm	generic	map
if_l4dm	generic	fault
if_l4dm	generic	close
if_l4dm	generic	close_all
if_l4dm	generic	share
if_l4dm	generic	revoke
if_l4dm	generic	check_rights
if_l4dm	generic	transfer
if_l4dm	generic	copy
if_l4dm	generic	set_name
if_l4dm	generic	get_name

Tabelle A.1: Ausgewertete Funktionsschnittstellen

Name	Interface	Funktion
if_l4dm	generic	show_ds
if_l4dm	generic	list
if_l4dm	mem	open
if_l4dm	mem	size
if_l4dm	mem	resize
if_l4dm	mem	phys_addr
if_l4dm	mem	is_contiguous
if_l4dm	mem	lock
if_l4dm	mem	unlock
if_l4dm	mem	info
if_l4dm	memphys	dmphys_open
if_l4dm	memphys	dmphys_copy
if_l4dm	memphys	dmphys_pagesize
if_l4dm	memphys	dmphys_poolsize
if_l4dm	memphys	dmphys_debug
l4exec	bin	open
l4exec	bin	ftype
l4exec	bin	close
l4exec	bin	link
l4exec	bin	get_symbols
l4exec	bin	get_lines
l4exec	bin	get_dsym
l4vfs	extendable	attach_namespace
l4vfs	extendable	detach_namespace
l4	io	register_client
l4	io	unregister_client
l4	io	map_info
l4	io	request_region
l4	io	release_region
l4	io	request_mem_region
l4	io	search_mem_region
l4	io	release_mem_region
l4	io	request_dma
l4	io	release_dma
l4	io	release_client
l4	io	pci_find_slot
l4	io	pci_find_device
l4	io	pci_find_class
l4	io	pci_enable_device
l4	io	pci_disable_device
l4	io	pci_set_master
l4	io	pci_set_power_state
l4	io	pci_read_config_byte
l4	io	pci_read_config_word
l4	io	pci_read_config_dword
l4	io	pci_write_config_byte
l4	io	pci_write_config_word
l4	io	pci_write_config_dword

Tabelle A.1: Ausgewertete Funktionsschnittstellen

Name	Interface	Funktion
l4	ts	allocate
l4	ts	create
l4	ts	free
l4	ts	kill
l4	ts	kill_recursive
l4	ts	owner
l4	ts	taskno_to_taskid
l4	ts	exit
l4	ts	dump
l4	ts	do_kill_reply
l4	rm	add
l4	rm	remove
l4loader	app	open
l4loader	app	cont
l4loader	app	kill
l4loader	app	dump
l4loader	app	info
l4loader	app	lib_open
l4loader	app	lib_link
l4loader	app	lib_dsym
l4vfs	mmap	mmap
l4vfs	mmap	msync
l4vfs	mmap	munmap
l4vfs	name_server	attach_object
l4vfs	name_space_provider	register_volume
l4vfs	name_space_provider	unregister_volume
names	names	register
names	names	register_thread
names	names	unregister_thread
names	names	unregister_task
names	names	query_name
names	names	query_id
names	names	query_nr
names	names	dump
l4vfs	net_io	accept
l4vfs	net_io	bind
l4vfs	net_io	connect
l4vfs	net_io	listen
l4vfs	net_io	recvfrom
l4vfs	net_io	recv
l4vfs	net_io	send
l4vfs	net_io	sendto
l4vfs	net_io	sendmsg
l4vfs	net_io	shutdown
l4vfs	net_io	socket
l4vfs	net_io	getsockname
l4vfs	net_io	setsockopt
l4vfs	net_io	socketpair

Tabelle A.1: Ausgewertete Funktionsschnittstellen

Name	**Interface**	**Funktion**
rmgr	rmgr	init_ping
rmgr	rmgr	set_small_space
rmgr	rmgr	set_prio
rmgr	rmgr	get_prio
rmgr	rmgr	get_task
rmgr	rmgr	free_task
rmgr	rmgr	free_task_all
rmgr	rmgr	task_new
rmgr	rmgr	get_irq
rmgr	rmgr	free_irq
rmgr	rmgr	free_irq_all
rmgr	rmgr	free_fpage
rmgr	rmgr	free_page
rmgr	rmgr	dump_mem
rmgr	rmgr	reserve_mem
rmgr	rmgr	free_mem_all
rmgr	rmgr	get_page0
rmgr	rmgr	get_task_id
rmgr	rmgr	set_task_id
rmgr	rmgr	privctrl
l4vfs	select_listener	send_notification
l4vfs	select_notify	request
l4vfs	select_notify	clear
signal	signal	kill
signal	signal	thread_kill
signal	signal	alarm
signal	signal	receive_signal
signal	signal	register_handler
signal	signal	unregister_handler
stream	io	push

Tabelle A.1: Ausgewertete Funktionsschnittstellen

Literaturverzeichnis

[App06] Apple Computer Inc.: *Kernel Programming Guide.* Apple Computer Inc., Oktober 2006.

[BALL90] Bershad, B.N., T.E. Anderson, E.D. Lazowska und H.M. Levy: *Lightweight Remote Procedure Call.* ACM Transactions on Computer Systems, 8(1):37–55, 1990.

[Cla04] Clauss, D.: *Implementation of the L4 Version x. 2 ABI in the Fiasco Microkernel.* Technischer Bericht, Technical report, TU Dresden, April 2004.

[Döb06] Döbel, Björn: *Request tracking in DROPS.* TU Dresden, Institut für Systemarchitektur, Mai 2006.

[Hoh98] Hohmuth, Michael: *The Fiasco Kernel: Requirements Definition.* Technischer Bericht TUD-FI-12, TU Dresden, Dezember 1998.

[Hoh02] Hohmuth, Michael: *The Fiasco Kernel: System Architecture.* Technischer Bericht TUD-FI02-06, TU Dresden, Dezember 2002.

[Kau05] Kauer, Bernhard: *L4. sec Implementation - Kernel Memory Management.* Diplomarbeit, Technische Universitat Dresden, 2005.

[KV05] Kauer, Bernhard und Marcus Völp: *L4.Sec Preliminary Microkernel Reference Manual, ver. 0.2.* TU Dresden, Institut für Systemarchitektur, Oktober 2005.

[Lie95] Liedtke, J.: *On micro-kernel construction.* ACM SIGOPS Operating Systems Review, 29(5):237–250, 1995.

[LIJ97] Liedtke, J., N. Islam und T. Jaeger: *Preventing denial-of-service attacks on a-kernel for WebOSes.* Proc. 6th HotOS, May, 1997.

[Loe92a] Loepere, K.: *Mach 3 Kernel Interfaces.* Open Software Foundation and Carnegie Mellon University, Nov, 1992.

[Loe92b] Loepere, K.: *Mach 3 Kernel Principles.* Open Software Foundation and Carnegie Mellon University, 1992.

[LRH01] Löser, Jork, Lars Reuther und Hermann Härtig: *A Streaming Interface for Real-Time Interprocess Communication.* Technischer Bericht TUD-FI01-09, TU Dresden, August 2001.

[Rit93] RITCHIE, D.S.: *The raven kernel: a microkernel for shared memory multiprocessors.* Diplomarbeit, University of British Columbia, 1993.

[RN93] RITCHIE, D.S. und G.W. NEUFELD: *User level IPC and device management in the Raven kernel.* USENIX Microkernels and Other Kernel Architectures Symposium, Seiten 111–126, 1993.

[SB90] SCHROEDER, M.D. und M. BURROWS: *Performance of the Firefly RPC.* ACM Transactions on Computer Systems (TOCS), 8(1):1–17, 1990.

[Sha99] SHAPIRO, J.S.: *EROS: A CAPABILITY SYSTEM.* Doktorarbeit, University of Pennsylvania, 1999.

[Sha03] SHAPIRO, J.S.: *Vulnerabilities in synchronous IPC designs.* Security and Privacy, 2003. Proceedings. 2003 Symposium on, Seiten 251–262, 2003.