Großer Beleg

Speicherverwaltung für den L4-Alpha Kern mit Echtzeitanforderungen

Volkmar Uhlig

volkmar@os.inf.tu-dresden.de

im Juni 98

Technische Universität Dresden
Fakultät Informatik, Lehrstuhl für Betriebssysteme

Kapitel 1 - Einführung

In Multitasking-Systemen werden gleichzeitig laufende Prozesse durch virtuelle Adreßräume voneinander getrennt und voreinander geschützt. Die Abbildung der virtuellen Adressen erfolgt dabei durch Seitentabellen, die im Kern geführt werden. Jeder Adreßraum hat seine eigene Seitentabelle, in der die Zugriffsrechte auf den Speicher vermerkt sind.

In monolithischen Betriebssystemen ist diese Speicherverwaltung vollständiger Bestandteil des Kerns.

Mikrokerne realisieren im Gegensatz dazu die Speicherverwaltung auf Nutzerebene. Die Nutzeranwendungen können mit Hilfe der Interprozeßkommunikation die Zugriffsrechte auf Speicherseiten weitergeben. Aufgabe des Kerns ist es dabei, diese Operationen sicher auszuführen und die Beziehungen zwischen den Adreßräumen zu speichern.

Die vorliegende Arbeit beschreibt grundlegende Verfahren zur mehrstufigen Seitenverwaltung in Adreßräumen mit nebenläufigen Prozessen. Die Datenbank wurde im Hinblick auf die Echtzeitunterstützung des L4-Kerns entworfen. Die Implementation wurde auf Basis des L4-Mikrokerns für DEC-Alpha von Sebastian Schönberg [Sch96] entwickelt.

Kapitel 2 geht auf den derzeitigen Stand der Technik ein und die Umsetzung in den gängigen Systemen. Kapitel 3 verdeutlicht den Entwurf der Speicherdatenbank und beschreibt die daraus resultierenden Algorithmen. Die in Kapitel 4 beschriebene Referenzimplementation für den Alpha-Prozessor beschäftigt sich detaillierter mit den beschriebenen Verfahren. Abschließend werden in Kapitel 5 die Meßergebnisse und das Zeitverhalten der Datenbankoperationen präsentiert.

Kapitel 2 - Stand der Technik

2.1 Virtueller Speicher

Fast alle Multitasking-Systeme sind mit mehreren virtuellen Adreßräumen realisiert, um die gleichzeitig laufenden Applikationen voreinander zu schützen. Die Umsetzung von virtueller Adresse auf die physische Speicheradresse wird mit Hilfe von Seitentabellen realisiert. Die meisten Systeme bieten außerdem die Möglichkeit der dynamischen Auslagerung von Speicher auf die Festplatte an, wenn der Systemspeicher nicht ausreicht.

Greift eine Anwendung auf ausgelagerte und somit nicht verfügbare Daten zu, kommt es zu einem Seitenfehler. Dieser wird vom Betriebssystem behandelt und die entsprechende Seite wird von der Festplatte eingelesen. Ist die virtuelle Adresse ungültig, d.h. es wurde kein physischer Speicher an dieser Stelle referenziert, wird im Normalfall die Anwendung vom Betriebssystem beendet.

In modernen Mikrokernen wird dieser Ein- und Auslagerungsalgorithmus als Nutzeranwendung realisiert. Beim Zugriff auf eine nicht vorhandene Seite lagert der Kern keine Seite ein, sondern generiert eine Nachricht an den zugewiesenen Manager (*user-level-pager*). Das setzt voraus, daß Nutzerapplikationen die Möglichkeit haben, Speicher an andere Anwendungen weiterzugeben und diesen auch wieder zu entziehen.

2.2 Gemeinsamer Zugriff auf Speicher

Der gemeinsame Zugriff zweier Applikationen auf den gleichen Speicher, die zum Beispiel aus Sicherheitsgründen in unterschiedlichen Adreßräumen ausgeführt werden, ist die schnellste Möglichkeit, Informationen zwischen den Anwendungen auszutauschen. Alle derzeit marktrelevanten Betriebssysteme bieten diese Möglichkeit als Kernfunktion an. Im Folgenden wird auf die Realisierung einiger Betriebssysteme kurz eingegangen.

2.2.1 Windows NT

Windows NT unterstützt gemeinsame Speichernutzung über *Shared Sections*. Dabei fordert eine Task explizit einen bestimmten Speicherbereich für gemeinsame Nutzung an. Dieser Bereich kann über eine Zugriffsnummer (Handle) oder über einen Namen referenziert werden. Applikationen, die auf diesen Bereich zugreifen wollen, müssen den freigegebenen Speicher explizit in ihren eigenen Adreßraum einblenden. Die Zugriffsrechte in den unterschiedlichen Adreßräumen werden dabei über den Objekt-Manager von Windows NT beschränkt, der die Zugriffsrechte auf alle Systemobjekte mit Hilfe von Zugriffssteuerlisten verwaltet.

2.2.2 UNIX/Linux

Linux realisiert den gemeinsamen Speicherzugriff indem mehrere Prozesse gleichzeitig die selbe Datei öffnen. Es besteht die Möglichkeit, eine geöffnete Datei linear im Adreßraum einzublenden, was in gemeinsam genutzten Speicher resultiert. Dieser Ansatz ist vergleichbar mit Windows NT.

2.3 Der L4-Mikrokern

Beide genannten Ansätze haben gemeinsam, daß die Zugriffsrechte durch Zugriffssteuerlisten geregelt werden.

Der L4-Kern realisiert die Beschränkung der Zugriffsrechte mit Hilfe der Interprozeßkommunikation (IPC). Wenn einem Adreßraum Zugriff auf einen physischen Speicherbereich gewährt werden soll, so wird dieser Speicher mit Hilfe einer Flexpage beschrieben und per IPC

die Zugriffsrechte übertragen. Der Kern trägt dabei die gesendeten Speicherseiten in die Seitentabelle des Empfängeradreßraumes ein. Flexpages sind detailliert in [HWL96] beschrieben.

2.3.1 Seitenfehlerbehandlung mit Hilfe der Interprozeßkommunikation

Die in monolithischen Betriebssystemen realisierten Paging-Verfahren setzen Festplattentreiber und Dateisysteme im Kern voraus. Um ähnliche Verfahren in mikrokern-basierten Systemen zu realisieren, werden Seitenfehler durch Nutzerapplikationen behandelt.

Der L4-Kern setzt bei einem Seitenfehler eine Nachricht auf, die der behandelnden Aktivität zugestellt wird. Jeder Aktivität wird dabei ihr spezieller Pager zugewiesen, der bei Seitenfehlern die Speicherseiten mittels Interprozeßkommunikation zustellt.

Dieser Ansatz setzt voraus, daß im Adreßraum des Pagers Speicher zur Verfügung steht. Deshalb existiert beim Systemstart von L4 ein initialer Adreßraum, Sigma 0, der den gesamten physischen Speicher virtuell eingeblendet hat. Sigma 0 ist außerdem der Pager für alle beim Systemstart erzeugten Nutzeraktivitäten.

Jede Aktivität hat die Berechtigung, vorhandenen Speicher an andere Adreßräume weiterzugeben. Dadurch entstehen hierarchische Vergaben von Zugriffsrechten, die vom Kern geführt werden müssen.

2.3.2 L4-Intel-Implementation

Die L4-Intel-Implementation speichert die hierarchische Vergabe der Seiten in Form eines Baumes. Dabei reserviert der Kern beim Starten des Systems einen Bereich fester Größe. Der Einstiegspunkt in den Baum wird über die physische Adresse der Speicherseite (bzw. das Sigma0-Mapping) generiert. Ausgehend von diesem Haupteintrag bildet der Baum die Vergabe ab.

Zum Aufsuchen des Eintrages für einen bestimmten Adreßraum muß im schlimmsten Fall der gesamte Baum durchsucht werden. Diese Operationen sind in der Originalimplementation mit gesperrten Unterbrechungen realisiert, was unter anderem zu starker Unterbrechungslatenz führt.

2.3.3 C++-Implementation des L4-Kerns

Die C++-Implementation des L4-Kerns von Michael Hohmuth wird die Vergabe von Zugriffsrechten auf Seiten in Form von Tabellen speichern. Für jede Kachel wird eine separate Tabelle angelegt. Die Organisation und Reihenfolge der Einträge in der Tabelle spiegeln die Beziehung zwischen den Adreßräumen wieder. Die Tabelleneinträge sind sehr klein und durch Verschiebung der Einträge soll zusätzlich die Cache-Nutzung optimiert werden. Der Nachteil dieses Ansatzes ist, daß bei Einfügen eines Eintrages gegebenenfalls die gesamte Tabelle verschoben werden muß, was bei großen Tabellen zu starker Systemlast führen kann.

Kapitel 3 - Entwurf

In diesem Kapitel werden der Aufbau und die Algorithmen der Speicherdatenbank beschrieben. Dabei ist der Schwerpunkt auf die Echtzeitfähigkeit des Entwurfs gesetzt.

3.1 Begriffsbestimmungen

- Eine Kachel ist ein physischer Speicherbereich, dessen Größe plattformabhängig ist.
- Eine Seite ist die Abbildung einer Kachel in einen Adreßraum an eine spezielle virtuelle Adresse.
- Als Quelladreßraum wird der Adreßraum bezeichnet, der den Zugriff auf eine Seite gewährt und als Zieladreßraum der Adreßraum, der Zugriffsrechte auf diese Seite gewährt bekommt.
- Ein Mapping ist ein Eintrag in der Datenbank. Er repräsentiert, daß in einem Adreßraum mit einer speziellen virtuellen Adresse eine Kachel angesprochen werden kann.

3.2 Ziel

Im L4-Kern werden Adreßräumen die Zugriffsrechte auf Speicherseiten mit Hilfe der Interprozeßkommunikation gewährt und durch einen speziellen Systemruf entzogen (siehe dazu 3.3.1). Dies geschieht in hierarchischer Form, so daß weitergegebene Rechte ebenfalls mit entzogen werden (Abbildung 1). Die vergebenen Zugriffsrechte und ihre Beziehungen untereinander müssen im System gespeichert werden. Dies erfolgt in Form einer Datenbank, die für jede Assoziation einer Kachel den jeweiligen Adreßraum und die Abhängigkeit zu den anderen Adreßräumen vermerkt.

Eine Kachel kann in einem Adreßraum mehrfach mit unterschiedlichen virtuellen Adressen eingetragen sein. Dies ist zum Beispiel möglich, wenn Adreßraum 1 an 2, 2 an 3 und 3 wieder an den Adreßraum 1 Zugriff auf die Seite gewährt. Wenn Adreßraum 2 die Seite entzieht, so muß die Seite aus Adreßraum 3 und 1 ebenfalls entfernt werden, jedoch ist an einer anderen virtuellen Adresse im Adreßraum 1 die Seite weiterhin verfügbar (Abbildung 1).

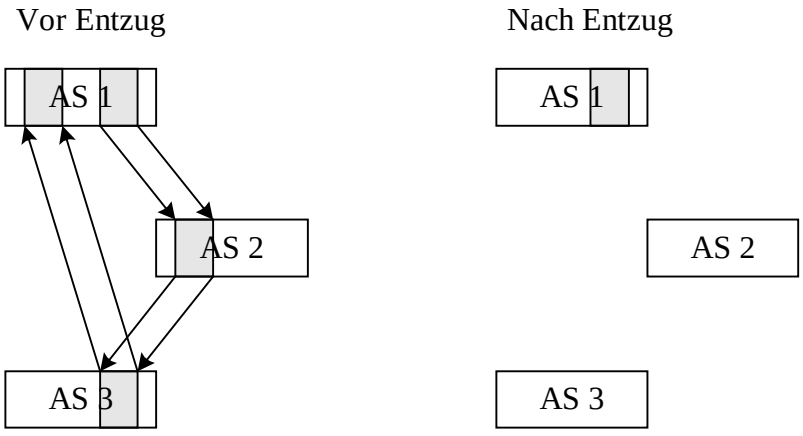

Abbildung 1: Hierarchische Vergabe und Entzug von Zugriffsrechten

3.3 Anforderung an die Datenbank

Der L4-Kern soll Basis für Echtzeitapplikationen, wie zum Beispiel *Video-on-Demand*, eingesetzt werden. Daraus entstehen harte Anforderungen an alle Systemfunktionen bezüglich

des Antwortzeitverhaltens, der Unterbrechungslatenz des Systems und der Speichernutzung und -verwaltung. Die Speicherdatenbank als fundamentaler Bestandteil der Speicherverwaltung des Kerns muß sich ebenfalls an die Vorgaben halten.

Folgende Randbedingungen wurden für den Entwurf festgelegt:

- **Speichernutzung:** Ein Kern sollte so wenig wie möglich Hauptspeicher für Verwaltungsaufgaben verwenden, damit der Speicher des Systems den Anwendungsprogrammen zur Verfügung steht. Da ein Datenbankeintrag pro virtueller Referenzierung auf eine Seite pro Adreßraum notwendig ist, muß versucht werden, die Datenbankstrukturen zu minimieren. Freigegebene Datenbankeinträge sind vom Kern wiederzuverwenden.
- **Echtzeit:** Um Echtzeitapplikationen nicht zu behindern, dürfen längere Operationen nicht systemblockierend sein (gesperrte Unterbrechungen). Das bedeutet, daß längere atomar auszuführende Abschnitte mit regulierenden Mechanismen, wie zum Beispiel kritischen Abschnitten, realisiert werden.
- **Zeitverhalten:** Das Erstellen eines Mappings muß ein determiniertes Zeitverhalten aufweisen, damit die Dauer einer Sende-Operation für Echtzeitapplikationen kalkulierbar ist.
- **Rekursion:** Die im Kernmodus ausgeführten Funktionen (Systemrufe) nutzen den Thread-spezifischen Kernstack. Da dieser in seiner Größe beschränkt ist, sind die Operationen der Mapping-Datenbank mit iterativen Algorithmen zu implementieren.
- **Konsistenz zwischen den Adreßräumen und der Datenbank:** Die Datenbank muß nach Rückkehr aus einer adreßraumverändernden Funktion den tatsächlichen Zustand der Adreßräume repräsentieren; das heißt, die Operation muß vollständig ausgeführt und beendet sein.
- **Unabhängigkeit der Einträge:** Die Modifikation eines Eintrages in der Datenbank darf nicht die Modifikation eines anderen, unabhängigen Eintrages blockieren. Dies soll selbst dann gewährleistet sein, wenn beide Einträge die gleiche Kachel referenzieren.
- **Prioritätsumkehr und „Verhungern“:** Da der L4-Kern mit harten Prioritäten arbeitet, sind mögliche Prioritätsinversionen durch den Entwurf der Datenbank auszuschließen.

3.3.1 Die L4-Systemrufe

Es wird hier nur auf die L4-Rufe eingegangen; die Adreßräume modifizieren. Eine vollständige Dokumentation liegt in [Lie96] vor.

An der Weitergabe von Zugriffsrechten auf eine Kachel zwischen zwei Adreßräumen müssen immer zwei Aktivitäten beteiligt sein - der Sender und der Empfänger; am Entzug von Rechten ist nur eine Aktivität beteiligt.

- **Grant:** Eine Aktivität kann jede Seite ihres Adreßraumes in einen anderen Adreßraum abgeben. Die Seite wird dabei aus dem Adreßraum des Senders entfernt und beim Empfänger an der vorgegebenen Adresse eingeblendet. Der Sender kann nur Seiten vergeben, auf die Zugriffsrechte bestehen. Die Seite im Zieladreßraum wird entweder mit gleichen oder reduzierten Zugriffsrechten angelegt.
- **Map:** Eine Aktivität kann die Zugriffsrechte auf jede Seite ihres Adreßraumes an einen anderen Adreßraum durch die Map-Operation weitergeben. Die Seite wird im Adreßraum des Empfängers an der vorgegebenen Adresse eingetragen. Der Sender hat die Möglichkeit, die Zugriffsrechte einzuschränken.
- **Flush.** Eine Aktivität kann die Zugriffsrechte auf jede Seite ihres Adreßraumes entfernen oder beschränken. Die Flush-Operation entfernt die Zugriffsrechte aus allen Adreßräumen, in denen die Seite direkt oder indirekt durch die Map-Operation verfügbar gemacht

wurde. Dabei kann die Seite des eigenen Adreßraums wahlweise mit eingeschlossen werden; Sigma 0 ist dabei als initialer Adreßraum ausgeschlossen.

- **Flush des gesamten Adreßraumes.** Der Besitzer einer Task kann diese löschen. Dabei werden alle im Adreßraum assoziierten Seiten entfernt und, identisch zur Flush-Operation, alle durch die Map-Operation weitergereichten Seiten ebenfalls entzogen.

3.3.2 Funktionen der Datenbank

Die L4-Systemrufe initiieren die Veränderungen an der Datenbank. Die Datenbank bietet folgende zu den L4-Rufen korrespondierende Schnittstelle:

- **`Add_Mapping`** stellt unter Angabe des Quell-Eintrags einen neuen Eintrag für den Zieladreßraum her.
- **`Grant_Mapping`** ändert die Adreßraumnummer und die virtuelle Adresse eines existierenden Eintrages.
- **`Flush_Mapping`** löscht alle Datenbank- und die korrespondierenden Seitentabelleneinträge in allen Adreßräumen, an welche die Seite durch eine Map-Operation direkt oder indirekt vergeben wurde. Dabei kann gewählt werden, ob die Seite im eigenen Adreßraum verbleiben, oder auch entfernt werden soll.
- **`Flush_Mapping_Read_Only`** reduziert die Zugriffsrechte auf die Seiten und die korrespondierenden Seitentabelleneinträge in allen Adreßräumen, an welche die Seite durch eine Map-Operation direkt oder indirekt vergeben wurde.
- **`Flush_Task`** führt `Flush_Mapping` für jede Seite des gewählten Adreßraumes durch.

3.3.3 Der Datenbank-Eintrag

Für jede virtuelle Adresse in einem Adreßraum, die eine Kachel referenziert und damit einen Seitentabelleneintrag hat, existiert in der Datenbank ein korrespondierender Eintrag. Mit Hilfe des Datenbankeintrages ist es möglich, alle direkt oder indirekt durch die Map-Operation in andere Adreßräume eingetragenen Seiten abzufragen.

3.3.4 Abzubildende Struktur durch die Mapping-Datenbank

Eine Seite in einem Adreßraum kann beliebig oft an andere Adreßräume weitergegeben werden (map), was eine 1 : n Beziehung ergibt. Diese wird in der Mapping-Datenbank mit Hilfe eines 1 : n Baumes abgebildet (siehe Abbildung 2).

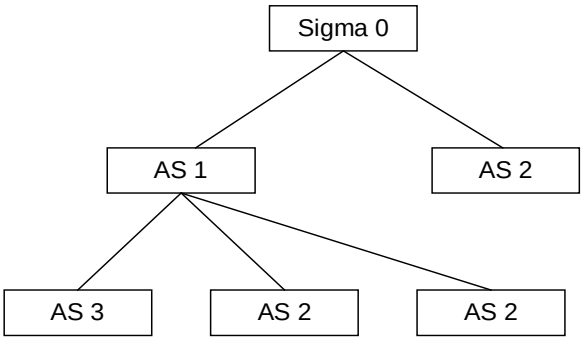

Abbildung 2: Beziehung zwischen den Adreßräumen für eine Seite

3.3.5 Modifikationen an der Datenbank

Die Struktur der Datenbank wird durch die zwei Operationen Einfügen und Löschen eines Eintrages verändert. Die Grant-Operation ändert lediglich den Inhalt eines Eintrages, aber nicht die Struktur.

Das Hinzufügen und Löschen eines Datenbankeintrages sind dabei atomare Operationen.

Durch die in 3.3 gestellten Anforderungen bezüglich der Unterstützung von Echtzeitapplikationen muß es möglich sein, von jedem Eintrag alle gehaltenen Referenzen direkt zu modifizieren, ohne daß zum Beispiel Listen durchlaufen werden. Deshalb wurden mehrere doppelt verkettete Listen in der Datenbank verwendet, obwohl die Größe der Einträge dadurch zunimmt.

3.3.6 Verkettung der Einträge

Um die Größe der Datenbankeinträge zu minimieren, werden alle Einträge, die vom gleichen Quell-Eintrag erzeugt wurden, in einer Liste verkettet. Diese ist, wie in 3.3.5 verdeutlicht, doppelt verkettet. Der Quelleintrag hält nur einen Verweis auf den Anfang der Liste, anstatt jeden einzelnen Eintrag separat zu referenzieren.

Um Rekursionen auszuschließen (siehe 3.3), ist von jedem Eintrag der zugehörige Quell-Eintrag referenzierbar. In der verketteten Liste verweisen der erste und letzte Eintrag zurück auf den Quell-Eintrag.

Um die L4-Operation `Flush_Task` effizient zu implementieren, werden alle Einträge eines Adreßraumes ebenfalls durch eine doppelt verkettete Liste miteinander verbunden. Damit erübrigt sich das vollständige Durchsuchen der Seitentabellen, was bei großen Adreßräumen, wie zum Beispiel dem 64 Bit Adreßraum des Alpha-Prozessors mit maximal 2^{30} Seiten, sehr lange dauern kann. Die Doppelverkettung dieser Liste ist für die `Grant_Mapping` Operation notwendig.

3.3.7 Inhalt der Datenbankeinträge

Zu den genannten Listen, die die Vergabe der Zugriffsrechte und Beziehungen zwischen den Adreßräumen widerspiegeln, werden in jedem Eintrag noch die Adreßraumnummer und die virtuelle Adresse der Seite gespeichert. Es reicht nicht aus, die Adresse der Kachel zu speichern, da eine Kachel an mehreren virtuellen Adressen im gleichen Adreßraum verfügbar sein kann.

Die Zugriffsrechte werden nicht im Datenbankeintrag geführt, da diese durch die Seitentabelle ermittelt werden können. Aus Geschwindigkeitsaspekten wäre das Speichern einer Referenz auf den Seitentabelleneintrag denkbar, wurde aber in der Alpha-Lösung nicht realisiert,

da die Adressen der Seitentabelleneinträge bei *Guarded-Pagetables* (beschrieben in [Sch96]) nicht statisch sind.

Beim Löschen einer Seite (Flush-Operation) muß deshalb erst die Adresse des Seitentabelleneintrags berechnet werden.

3.4 Entwurfsansätze

3.4.1 Konsistenz und Konkurrenz

Um die in 3.3 genannten Echtzeitanforderungen mit nur kurzen, nicht unterbrechbaren kritischen Abschnitten zu realisieren, müssen Sperrmechanismen in die Datenbank integriert werden.

Das Hinzufügen von neuen Einträgen in die Datenbank kann mit atomaren, nicht unterbrechbaren Funktionen realisiert werden, da diese Operationen sich auf wenige Umkettungen beschränken und damit ein deterministisches Zeitverhalten aufweisen.

Im Gegensatz dazu muß das Löschen eines Eintrages und der damit betroffenen Einträge in anderen Adreßräumen mit offenen Unterbrechungen realisiert werden.

Die Sperrung innerhalb der Datenbank ist auf drei Ebenen möglich. Die umfassendste ist die Sperrung der gesamten Datenbank, solange eine Löschoperation (`Flush`) durchgeführt wird.

Dies ist aber gesperrten Unterbrechungen gleichzustellen, da in diesem Fall von keinem Adreßraum Seiten weitergegeben werden können.

Alternativ ist die Sperrung auf Kachelebene möglich. Jedoch können dann Applikationen mit Zugriffsrechten auf eine Kachel andere Anwendungen behindern, die ebenfalls Operationen über der Kachel ausführen, was jedoch im Konflikt zu den Anforderungen aus 3.3 steht. Dort wird die Unabhängigkeit zweier Aktivitäten bei Modifikation der Zugriffsrechte auf die gleiche Kachel gefordert. Außerdem bietet diese Realisierung einen Ansatzpunkt für *Denial-of-Service* Attacken, da es möglich ist, Echtzeitanwendungen durch Nichtechtzeitanwendungen zu blockieren.

Die dritte und auch realisierte Variante ist die Sperrung auf Eintragsebene der Datenbank. Damit können Echtzeitanwendungen die Seite an andere Adreßräume weitergeben, ohne daß das Zeitverhalten sich ändert. Das Löschen von Einträgen ist aufgrund der nicht bekannten Anzahl von betroffenen Einträgen keine zeitdeterministische Operation. Falls Echtzeitanwendungen Seiten entfernen müssen, dann kann dies nur durch eine zweite Aktivität realisiert werden, die aber nicht vorhersagbar lange läuft. Unter diesem Gesichtspunkt hat die Sperrung eines Eintrag von einer Nichtechtzeitanwendung keine behindernden Auswirkungen auf die Echtzeitapplikation. Allerdings darf die Sperrung eines Eintrages nicht zu einer *Deadlock* Situation oder Prioritätsinversion führen.

Im folgenden werden zwei Lösungsansätze für die Datenbank diskutiert. Der erste Ansatz ist nicht realisierbar, hat aber zur endgültigen Lösung beigetragen, und wird deshalb kurz beschrieben.

3.4.2 Abtrennen eines Teilbaumes

Der Grundgedanke dieses Ansatzes basiert darauf, daß Löschoperationen auf der gleichen Zielmenge nicht doppelt ausgeführt werden. Wenn eine Aktivität einen Teilbaum in der Datenbank löscht, dann sperrt sie den obersten Eintrag des Teilbaumes. Eine zweite Aktivität, die ebenfalls diese Einträge der Datenbank löscht, kettet den gesperrten Eintrag aus, markiert ihn als ungültig und setzt die angefangene Operation fort (Abbildung 3). Die sperrende Aktivität entsperrt den Eintrag nicht, sondern entfernt ihn. Damit ist die Konsistenz der Datenbank gesichert und niederpriore Aktivitäten können hochpriore nicht behindern.

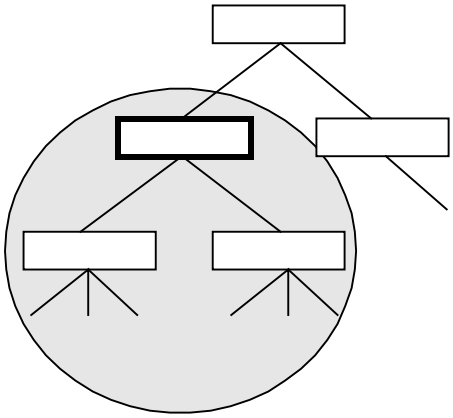

Abbildung 3: Löschen durch Abtrennung eines Teilbaumes

Der Ansatz hat sich aber als falsch herausgestellt, da die in 3.3 geforderte Adreßraumkonsistenz nicht gewährleistet ist. Es ist möglich, daß die von einer Aktivität aufgerufene Funktion bereits zurückkehrt, obwohl noch nicht alle Datenbankeinträge und damit auch die Seitentabelleneinträge entfernt sind. Damit kann die Seite, obwohl sie eigentlich bereits entzogen wurde, noch von anderen Aktivitäten verändert werden.

3.4.3 Warten auf Beendigung

Beim Löschen eines Eintrages und seiner Untereinträge wird in diesem Ansatz auch eine Sperrung des Eintrages vorgenommen. Wenn eine zweite Aktivität diesen Datenbankeintrag löschen will, dann muß diese Aktivität auf die Beendigung der noch laufenden Operation warten. Dieser Fall ist aber nur in zwei Situationen möglich; entweder haben beide Aktivitäten die gleiche oder die wartende Aktivität hat eine höhere Priorität. Der Ansatz des *busy waiting* ist deshalb, abgesehen von der Verschwendung von Systemressourcen, nicht realisierbar. Die blockierte Aktivität schaltet deshalb auf die sperrende Aktivität um, damit diese die Sperrung so schnell wie möglich wieder aufhebt (Abbildung 4).

Der Vorteil ist, daß auf diese Weise bei mehrfachen Sperrungen die Prioritäten mehrfach vererbt werden.

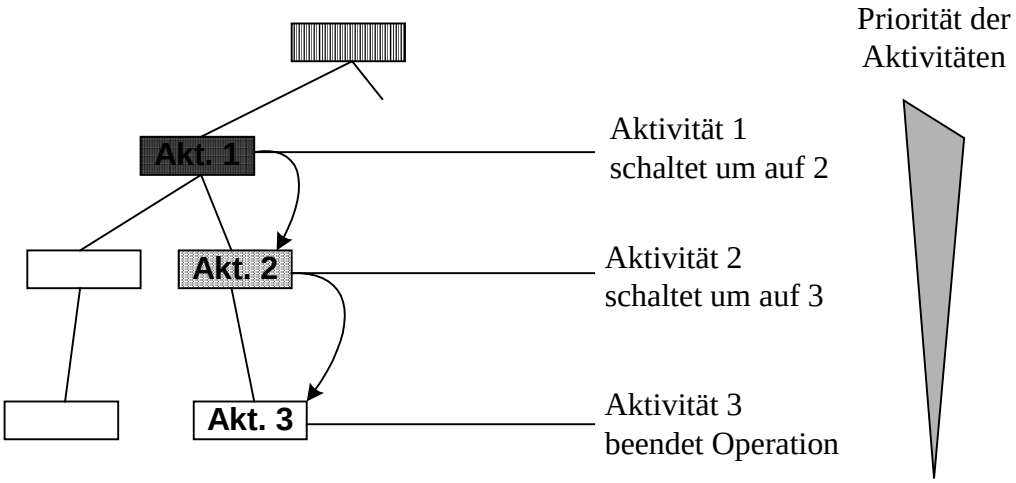

Abbildung 4: Prioritätsvererbung bei gesperrten Datenbankeinträgen

Beim Löschen von Einträgen werden diese unter normalen Umständen sofort zur Wiederverwendung freigegeben. Wenn andere Aktivitäten diesen Eintrag ebenfalls löschen wollen, dann darf der Eintrag nicht entfernt werden. Wenn der Eintrag sofort ausgekettet wird, dann entsteht die in Abbildung 5 dargestellte Situation, bei der die Struktur für andere wartende Aktivitäten zerstört wurde und deren Operation nicht mehr vollständig ausführbar ist.

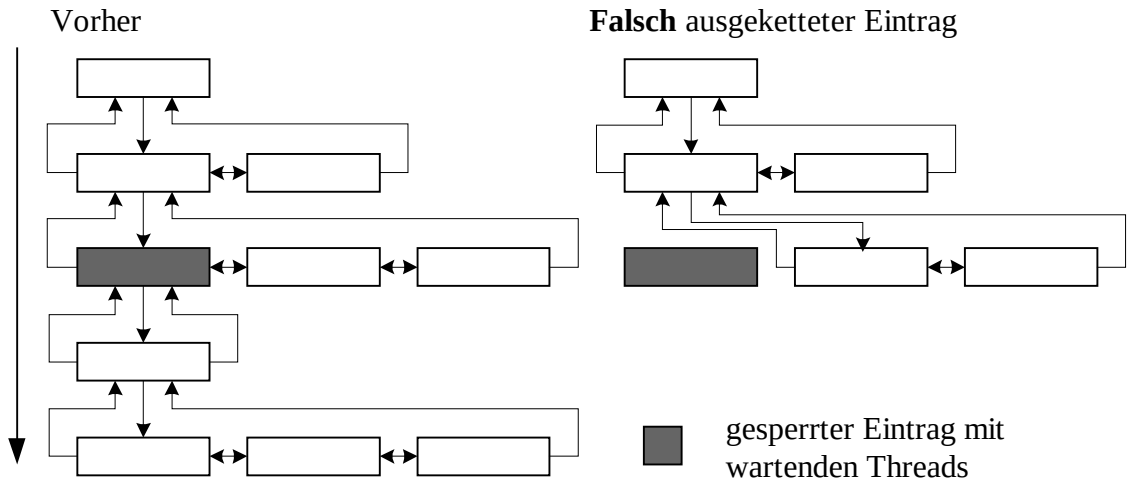

Abbildung 5: Falsche Auskettung bewirkt Inkonsistenz der Datenbank

Deshalb wird der Eintrag in der Datenbank belassen und erst dann freigegeben, wenn keine weitere Aktivität diesen Datenbankeintrag bearbeitet. Der Datenbankeintrag enthält einen Referenzzähler der wartenden Aktivitäten. Die Aktivität, welche die letzte Referenz freigibt, entfernt den Eintrag aus der Datenbank.

Das Hinzufügen von neuen Einträgen ist trotz einer laufenden Flush-Operation möglich. Abbildung 6 zeigt die Strukturen der Datenbank am Beispiel. Dabei werden die neuen Einträge nur angelegt, wenn der Quelleintrag nicht als ungültig markiert ist.

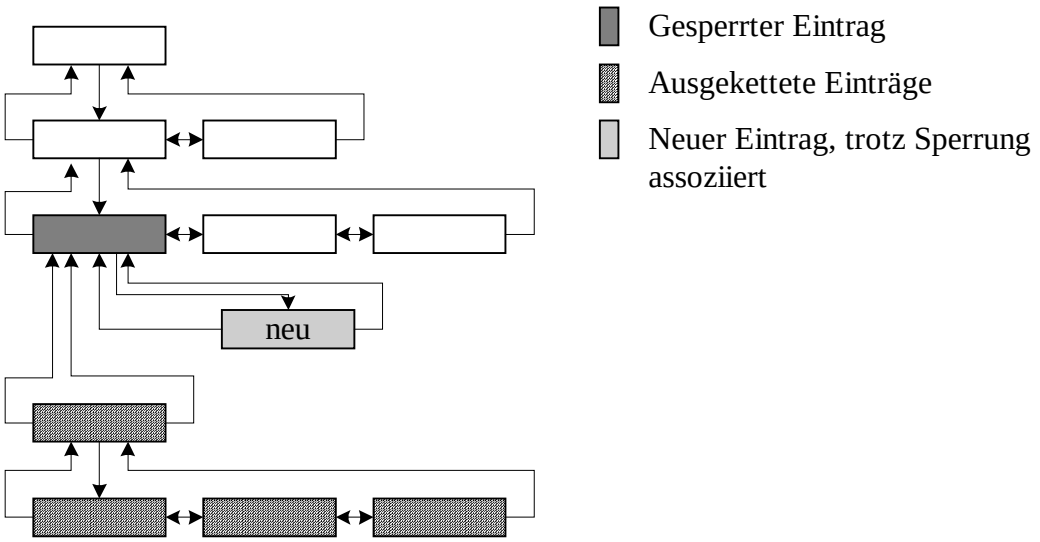

Abbildung 6: Assoziation eines neuen Mappings bei laufendem Flush

Kapitel 4 - Implementation

4.1 Alpha 21164 Plattform

Die realisierte Mapping-Datenbank ist für den DEC-Alpha-Prozessor 21164 implementiert worden.

Der Prozessor bietet drei relevante Prozessormodi an, den Nutzer- und Kernmodus sowie den PALmode (Privileged Architecture Library Mode).

Unterbrechungen, Ausnahmen und Maschinenfehler werden im nicht unterbrechbaren PALcode behandelt. Hier hat man Zugriff auf den physischen Speicher und *Memory-Mapped I/O* Geräte. Kernoperationen, wie die der Mapping-Datenbank, werden im unterbrechbaren Kernmodus ausgeführt. Für atomare Operationen besteht die Möglichkeit, Unterbrechungen zu verbieten.

4.1.1 Adreßraum

Der Alpha-Prozessor ist ein 64-Bit Prozessor. Eine Virtuelle Adresse wird durch eine vorzeichenlose 64 Bit Zahl dargestellt. Es sind die vier Seitengrößen 8kB, 64kB, 512kB und 2MB wählbar, woraus Adreßräume mit 43, 47, 51 und 55 Bit resultieren. Der L4-Kern arbeitet mit 8 kByte Seitengröße und damit dem 43 Bit Adreßraum. Bei Referenzierung einer Adresse müssen die Bits 63 bis 43 identisch sein. Damit ist der Bereich gültiger Adressen von 0 bis 0000.03FF.FFFF.FFFF und FFFF.FC00.0000.0000 bis FFFF.FFFF.FFFF.FFFF. Der L4 Kern nutzt den ersten Bereich für Nutzeradreßräume und den zweiten für Kernadressen.

Im Fall eines *TLB-Misses* wird eine Ausnahmeroutine im PALmode angesprungen, die den TLB-Eintrag aus der Seitentabelle ausliest und einträgt.

Dies steht im Gegensatz zu anderen Architekturen, wie zum Beispiel der Intel Familie ab 386, bei der die Umsetzung in Form von Hardwareimplementation gelöst ist und damit feste Strukturen der Seitentabellen vorgegeben sind.

Der L4-Kern nutzt *Guarded-Page-Tables* für die Umsetzung der virtuellen Adressen, die eine effizientere Speichernutzung und bei spärlicher Adreßraumbesetzung auch wesentlich bessere Performance bieten. Insgesamt existieren pro Adreßraum 2^{30} adressierbare Seiten.

4.1.2 Gewählter Adreßbereich

Die Daten der Mapping Datenbank liegen im Kernspeicherbereich. Dieser wird virtuell adressiert und liegt oberhalb FFFF.FC00.0000.0000. Die eigentlichen Mapping-Einträge liegen im Bereich von FFFF.FFFF.D000.0000 bis FFFF.FFFF.DFFF.FFFF. Jeder Eintrag hat eine Größe von 32 Byte, deren genaue Struktur in 4.1.3 beschrieben wird. Damit ist es theoretisch möglich über 8 Millionen Mappings zu assoziieren, was aber 250 MB für Mappings reservierten Kernspeicher voraussetzen würde.

Der gewählte Speicherbereich hat einen weiteren Vorteil. Der Alpha Prozessor basiert auf einer LOAD/STORE Architektur. Referenzierte Adressen müssen in einem Register stehen.

Die Adressen der Mapping-Datenbank sind in den oberen 32 Bit identisch, nämlich 1 binär bzw. FFFF.FFFF.xxxx.xxxx. Um Speicherplatz zu sparen, können die oberen 32 Bit weggelassen werden. Diese müssen aber vor dem Zugriff auf die Speicheradresse wieder reproduziert werden. Normalerweise sind dazu mehrere Assembler Befehle notwendig, da keine 64 Bit Werte direkt in ein Register geladen werden können.

Der Alpha-Prozessor bietet den Befehl `LDL d_reg, offset(s_reg)` an, der in `d_reg` den Inhalt des Speichers an der virtuellen Adresse `s_reg` um `offset` verschoben, einliest. Dabei wird der eingelesene 32 Bit Wert auf 64 Bit vorzeichenerweitert. Der Wert D000.0000 wird

dabei in FFFF.FFFF.D000.0000 umgewandelt. Somit kann mit nur einem Maschinenbefehl aus einem 32 Bit Wert die vollständige 64 Bit Adresse generiert werden.

4.1.3 Struktur eines Mapping-Eintrages

Die in Kapitel 3 beschriebenen Anforderungen an die Mapping-Datenbank haben auf die folgende Struktur von Mapping-Einträgen für die Alpha-Implementation geführt:

<table>
<tr><td>Anzahl wartende Threads - 14 Bit</td><td>ID sperrender Thread, 18 bit</td><td colspan="2">Adresse, Sub-Mappings 32 Bit</td></tr>
<tr><td>Task ID 10 Bit</td><td colspan="2">Virtuelle Adresse – 47 Bit</td><td>Ungültig
1 Bit</td></tr>
<tr><td colspan="2">Adresse, Vorgänger pro Task - 32 Bit</td><td colspan="2">Adresse, Nachfolger pro Task - 32 Bit</td></tr>
<tr><td colspan="2">Adresse, Vorgänger - 32 Bit</td><td colspan="2">Adresse Nachfolger - 32 Bit</td></tr>
</table>

Dabei haben die Einträge folgende Bedeutung:

- Die Zeiger Vorgänger und Nachfolger verketten alle Einträge der Adreßräume, die eine Kachel vom selben Quelladreßraum mit Hilfe der Map-Operation eingeblendet bekommen haben. Die Adresse ist beim ersten und letzten Eintrag markiert, welche dann den übergeordneten Eintrag referenzieren.
- Die Zeiger Vorgänger und Nachfolger pro Task verketten alle Mappings die zum gleichen Adreßraum gehören. Der letzte Eintrag der Kette ist leer. Der Kopf dieser Kette wird in einer Indextabelle im *Task-Control-Block* des L4-Kernes geführt. Wenn der vorderste Eintrag der Liste ausgekettet wird, dann müßte der Indexeintrag mit Hilfe der Task-ID berechnet werden. Dies wird umgangen, indem der erste Eintrag auf die Indextabelle verweist.
- Der Zeiger Sub-Mappings ist leer, wenn keine Seiten durch die Map-Operation weitergegeben wurden, oder referenziert den ersten Eintrag in der Sub-Mapping-Liste.
- Die Task-ID (L4) ist die Adreßraum Nummer.
- Die virtuelle Adresse speichert mit 48 Bit die Anfangsadresse, an der die Kachel im Adreßraum eingeblendet wird. Diese ist ausgerichtet auf 8 kByte, und damit sind die untersten 13 Bit 0. Dieser Eintrag wird auch dazu verwendet, um den Eintrag als ungültig zu markieren. Dazu wird das unterste Bit auf 1 gesetzt. Alle Adressen eines Nutzeradreßraumes sind zwischen 0 und 0000.03FF.FFFF.FFFF. Deshalb werden nur 43 Bit benötigt, der Eintrag wird aber aus Effizienzgründen auf 48 Bit erweitert.
- ID des sperrenden Threads speichert die ID des Threads, der eine Flush-Operation auf diesem Mapping und dem darunterliegenden Mapping-Baum ausführt. Dieses Feld ermöglicht die Vererbung der Zeitscheibe an die sperrende Aktivität.
- Die Anzahl wartender Threads wird im gleichnamigen Feld gespeichert, wobei davon ausgegangen wird, das nicht mehr als 2^{14} Threads gleichzeitig warten. Falls dieser Fall doch einmal eintreten sollte, wird derzeit in eine Ausnahme-Behandlung verzweigt.

Das unterste Bit wird deshalb zur Markierung genutzt, da es von keinem gültigen Zeiger der Mapping-Datenbank gesetzt werden kann. Der Alpha-Befehlssatz enthält zwei Sprung-

Befehle, die entweder das unterste bzw. oberste Bit eines Register als Sprungbedingung nutzen, wodurch sehr effizienter Code zur Auswertung möglich ist:

```
BLBC/BLBS reg_s, label ⇨ Branch if lowest bit clear/set
BLT/BGET reg_s, label ⇨ Branch if less than zero
```

4.1.4 Referenzierung der Mappings

Die Referenzierung eines Datenbankeintrages bei gegebener Adresse der Seite und der Adreßraumnummer ist bei der Intel-Implementation von J. Liedtke eine der teuersten Operationen, da der gesamte Baum linear durchsucht wird. Da beim L4-Alpha Kern die Seitentabellen nicht in Hardware implementiert sind, konnten freie Bereiche der Seitentabellen zur Referenzierung genutzt werden. Die TLB-Einträge (*Translation Look Aside Buffer*) enthalten mehrere freie Bits, die die vollständige Mappingadresse (es werden genau 23 zur eindeutigen Bestimmung benötigt) bilden können. Die Anzahl der freien Bits beschränkt auch die Gesamtzahl möglicher Mappings im System. Da das Referenzieren eines Mappings im Verhältnis zu den auftretender *TLB-Misses* gering ist, wurde darauf geachtet, daß die Mapping-Bits mit möglichst wenigen und billigen Operationen ausmaskiert werden können.

Die Nutzung der Seitentabelle hat den zweiten Vorteil, daß Mappings nur dann referenzierbar sind, wenn der dazugehörige Seitentabelleneintrag vorhanden ist. Somit sind der Mappingbaum und die Seitentabelle automatisch konsistent.

4.1.5 Freier Mapping-Pool

Zum Mappen einer Seite wird ein freier Eintrag allokiert. Wieder freigegebene Mappings werden in einer einfach verkettete Liste geführt.

Die erste Implementation sah vor, daß immer dann, wenn kein Mapping mehr in der Freispeicherliste verfügbar ist, so viele Mappings generiert werden, wie in eine Seite (8 kByte) passen. Dieser Ansatz hat den Vorteil, daß für das Anfordern eines Mappings immer der gleiche Algorithmus anzuwenden ist. Die neuen Mappings werden in die Liste der durch die Flush-Operationen frei gewordenen Einträge eingekettet.

Das Zeitverhalten dieser Implementation ist jedoch für Echtzeitapplikationen ungeeignet, da die Allokation der 256 Einträge 20.000 Takte, bzw. 79µs benötigt.

Deshalb wurde auf eine andere Realisierung ausgewichen. Statt die gesamte Seite zu initialisieren und in einer Liste zu verketten, wird lediglich die Adresse des ersten ungenutzten Eintrages der Seite geführt, welcher linear weitergezählt wird. Wenn eine Seitengrenze erreicht wird, referenziert der Kern eine nicht vorhandene Kernseite und erzeugt einen *Kernel-Pagefault*. Dieser wird behandelt und es wird eine freie Kernseite an diese Adresse eingetragen. Jede Seite wird demzufolge vollständig mit Mappings gefüllt, ohne daß eine vorherige vollständige Initialisierung notwendig ist.

Beim Anfordern von freien Einträgen werden bevorzugt freigegebene Einträge benutzt, und erst dann eine Neuallokation durchgeführt. Diese Implementation hat zur Zeit noch den Nachteil, daß einmal angeforderte Seiten nicht wieder an den Kern zurückgegeben werden können. Wenn beim Start eines Systems viel Speicher allokiert wird, danach aber wieder freigegeben wird, so generiert der Kern sehr viele Datenbankeinträge. Diese sind dann in der Freispeicherliste, werden aber nicht wieder genutzt.

Es wäre möglich den ersten Eintrag pro Seite zur Referenzzählung der benutzten Einträge zu verwenden und mit einem *Garbage-Collector* vollständig ungenutzte Seiten wieder freizugeben. Außerdem ist es möglich, auf spärlich genutzten Seiten die Mappings umzuketten, was aber gegebenenfalls Auswirkungen auf Echtzeitanwendungen haben kann, da diese Operation atomar ausgeführt werden muß.

4.1.6 Realisierung der Mapping-Datenbank Funktionen

4.1.6.1 Add Mapping

Wie bereits beschrieben, läuft die Map-Operation vollständig atomar mit gesperrten Unterbrechungen ab. Zuerst wird ein freier Eintrag allokiert. Wenn das in 4.1.5 beschriebene Verfahren angewandt wird, ist es möglich, das ein Seitenfehler erzeugt wird. Die Pagefault Behandlungsroutine kehrt mit offenen Unterbrechungen zurück, womit die Operation unterbrechbar wird und die Mapping-Datenbank in einen inkonsistenten Zustand kommen kann. Deshalb wird beim Austritt aus der vollständig abgeschlossenen Map-Operation immer auf die nächste freie Adresse lesend zugegriffen und der Seitenfehler bewußt vorab generiert.

Die Map-Operation kann drei verschiedene Zustände des Quellmappings vorfinden.

- Mapping ist nicht gesperrt – der normale Fall:
  Die Map-Operation kettet den neuen Mapping-Eintrag als ersten Sub-Eintrag ein, speichert die Referenz im TLB-Eintrag der Seitentabelle und aktiviert die Unterbrechungen.
- Mapping-Eintrag ist gesperrt und die virtuelle Adresse ist gültig.
  Dieser Zustand kann entweder durch ein *Flush-Read-Only* oder durch ein Flush ohne eigenen Adreßraum entstehen. Die Art der Flush-Operation wird im Mapping-Eintrag geführt. Beim *Flush-Read-Only* dürfen keine neuen Mappings mit Schreibberechtigung generiert werden, da sonst ein Entzug von Zugriffsrechten nicht gesichert ist. Die entsprechenden Zugriffsrechte werden im TLB-Eintrag der Seitentabelle des Zieladreßraumes gleich mit modifiziert, wenn die Mapping-Referenz eingetragen wird.
- Der Mapping Eintrag ist gesperrt und die virtuelle Adresse ist ungültig.
  Dieser Fall tritt ein, wenn der Datenbankeintrag einschließlich des eigenen Adreßraumes gelöscht wird. In diesem Fall wird das neue Mapping nicht generiert und die Routine kehrt mit einem Fehlercode zurück.

4.1.6.2 Flush ohne eigenen Adreßraum

Diese Funktion löscht alle Mappings und die Einträge in den Seitentabellen. Dazu wird der Baum vollständig durchlaufen, jedoch werden alle Unterreferenzen sofort gelöscht. Die virtuelle Adresse wird vorher auf ungültig gesetzt, so das keine neuen Mappings assoziiert werden können. Durch das Löschen des Zeigers auf den ersten Untereintrag können sofort wieder neue Mappings vom obersten Eintrag assoziiert werden können. Beim zurückkehrenden Baumaufstieg sind dabei keine Sonderbehandlungen notwendig. Der oberste Eintrag wird nach Beendigung der Operation entsperrt.

Wenn andere Aktivitäten auf die Freigabe eines Eintrages in der Mitte des Baumes warten, so ist dies nur möglich, wenn deren zu löschender Bereich an diesem Eintrag beginnt. Anderenfalls hätten sie das darüberliegende bereits gesperrte Mapping sperren müssen, was nicht möglich ist. Gesperrte Mapping-Einträge werden dann einfach vollständig ausgekettet und die Vorgänger- und Nachfolgeradresse auf ungültig gesetzt. Ein Ausketten hat dann keine Auswirkungen auf den Baum und die wartenden Threads finden einen konsistenten Baum mit genau einem Eintrag vor. Dieser Fall ist in Abbildung 7 dargestellt, in der das Mapping im Adreßraum 2 von mehreren Thread modifiziert werden soll. Wenn die bearbeitende Aktivität die Sperrung aufhebt, ist der Eintrag bereits vollständig ausgekettet und als ungültig markiert. Die letzte wartende Aktivität gibt den Eintrag zur Wiederverwendung frei.

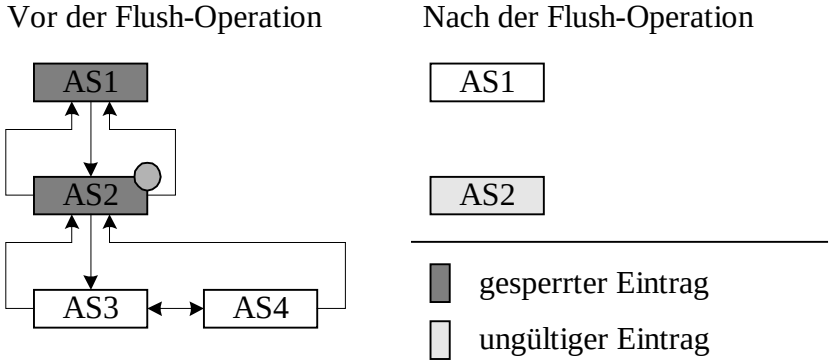

Abbildung 7: Flush-Operation mit mehreren wartenden Threads

4.1.6.3 Flush einschließlich des eigenen Adreßraumes

Bis auf das Entfernen des obersten Eintrages unterscheidet sich diese Funktion nicht von der vorhergehenden. Der oberste Eintrag wird allerdings auf ungültig gesetzt und beim Beenden der Funktion aus dem Baum entfernt. Wenn andere Threads auf das Freiwerden des Mappings warten, wird der Eintrag im Baum belassen und der letzte wartende Thread entfernt ihn. Alle frei werdenden Mappings werden bei den beiden Flush-Routinen in Form einer Ringliste abgelegt und wenn der gesamte Baum durchlaufen wurde, zur Freispeicherliste hinzugefügt. Dazu wird die Ringliste an einer Stelle aufgetrennt und an die bereits existierende Freispeicherliste vorn angefügt.

Beim Austragen der Mappings müssen die Einträge ebenfalls aus der Adreßraumliste ausgetragen werden.

4.1.6.4 Grant Mapping

Die Grant Funktion ist die einfachste und zugleich schnellste Operation. Es wird die Task-ID und die virtuelle Adresse verändert, sowie das Mapping in den neuen Seitentabelleneintrag hinzugefügt. Abschließend wird der Eintrag in die neue Task-Mapping-Liste umgetragen.

4.2 Besonderheiten der L4 Alpha Implementation

Beim Systemstart von L4 wird in den Adreßraum Sigma 0 der komplette physische Adreßraum, bis auf die vom Kern belegten Seiten, virtuell eingeblendet. Da die Behandlung von TLB-Fehlern und das Auswerten der Seitentabellen Bestandteil des Kerns sind, existieren für Sigma 0 keine Seitentabellen. Die TLB-Einträge, die normalerweise in der Seitentabelle gespeichert sind, können für Sigma 0 „on the fly“ generiert werden. Sigma 0 hat immer volle oder keine Zugriffsrechte auf eine Seite.

Diese Vorgehensweise spart viel Kernspeicher, da 2^{30} Kacheln mit 2^{30} dazugehörigen Mapping-Einträgen nicht generiert werden.

Da aber die Seitentabelle Voraussetzung für den Einstieg in die Mapping-Datenbank ist, existieren keine Mappings für Sigma 0, damit aber auch kein Wurzeleintrag für den Baum. Wenn eine Seite mehrfach von Sigma 0 aus an andere Adreßräume weitergegeben wird, entstehen ebenso viele Mapping-Bäume für die gleiche Kachel.

Leider schränkt diese Implementation die Funktionalität in zweierlei Hinsicht ein.

Sigma 0 kann durch die nicht vorhandenen Seitentabellen keine Seiten aus dem eigenen Adreßraum weggeben, da diese nicht verzeichnet werden können. Diese Einschränkung kann

durch eine Sonderbehandlung dieser Seiten umgangen werden. Dabei werden in den Seitentabellen von Sigma 0 die Seiten, auf die kein Zugriff besteht, eingetragen.

Das zweite Problem ist, daß aufgrund des nicht existierenden Wurzeleintrages für Sigma 0 nicht die Möglichkeit besteht, eine Seite sicher zu entziehen. Da aber den darunter liegenden Pagern nicht vertraut werden kann, sollte diese Möglichkeit existieren.

Als Lösungsansatz kann ein neuer Pager implementiert werden, der identisches Verhalten wie Sigma 0 hat, aber nur benutzte Seiten in seinem Adreßraum eingeblendet bekommt (und nicht den kompletten IO-Space). Dieser besitzt dann einen Wurzeleintrag und ermöglicht das sichere Entziehen von Zugriffsrechten.

Alternativ ist es möglich, für Sigma 0 die Seitentabelle *on-demand* zu generieren, und nur weitergegebene Seiten einzutragen.

4.3 Denial-of-Service Angriffe

Grundlegend sind mehrere *Denial-of-Service* Angriffe denkbar. Einige sind durch den Entwurf der Mapping-Datenbank ausgeschlossen, werden aber trotzdem beschrieben, um Entscheidungen über die entgültige Realisierung zu verdeutlichen.

4.3.1 Belegen des gesamten Kern-Speichers

Da jede referenzierbare Seite in einem Adreßraum ein Mapping in der Datenbank erzeugt, ist der offensichtlichste Angriffspunkt des Systems, alle Kernseiten aufzubrauchen. Damit sind keine neuen Mappings mehr assoziierbar und andere Tasks können keine weiteren Seiten in ihren Adreßraum eingetragen bekommen.

Diese Attacke ist zum Beispiel denkbar, wenn zwei Aktivitäten sich gegenseitig eine Kachel im Adreßraum an unterschiedliche virtuelle Adressen mehrfach zusenden. Jede Seite verbraucht ein Mapping und nach endlich langer Zeit ist der verfügbare Speicher aufgebraucht.

Diesem Angriff kann man begegnen, indem die Anzahl der Map-Operationen einer Seite beschränkt wird. Jedoch führt diese Lösung auf eine neuen Angriffspunkt, da dann eine Task die Seite durch „unerlaubtes“ Versenden für eine andere Task blockieren kann.

Die andere Variante ist, daß jede Task nur eine bestimmte Anzahl von Mappings nutzen darf. Alle weiteren Map-Operationen werden abgewiesen. Dies erzeugt aber einen enorm hohen Verwaltungsaufwand, da bei allen Operationen der Mapping-Datenbank diese Information mitgeführt und gegebenenfalls umgebucht werden müssen (Grant-Operation).

Eine weitere Möglichkeit ist die vorherige Reservierung von Einträgen, was aber eine Veränderung der L4-Systemschnittstelle zur Folge hat.

4.3.2 Problem der Prioritätsumkehr und des Aushungerns

Die in 3.4.3 beschriebene Variante zur Zeitscheibenvererbung an den sperrenden Thread stellt gleichzeitig eine Möglichkeit zum Blockieren eines Systems dar.

Folgender Fall ist konstruierbar:

Bei einem tiefen Mappingbaum ist jedes Mapping durch einen Thread gesperrt, wobei das unterste Mapping von dem Thread niedrigster Priorität gesperrt ist. Die Prioritäten der anderen Threads sind nach oben zunehmend. (siehe Tabelle 1).

Thread 1 Prio 99	Thread 2 Prio 98	Thread 3 Prio 97	Thread 4 Prio 96	...	Thread 97 Prio 3	Thread 98 Prio 2	Thread 99 Prio 1

Tabelle 1

Da Thread 1 die höchste Priorität hat, wird er ständig vom Scheduler aktiviert. Er gibt die Zeitscheibe an Thread 2 und so weiter. Dabei muß bei jedem Wechsel auf den anderen Thread-Kontext umgeschaltet werden. Wird aber die gesamte Zeitscheibe für Kontextwechsel aufgebraucht, so kann Thread 99 die Sperrung nicht aufheben und das System befindet sich im *Deadlock*.

Kapitel 5 - Performance Messungen

Die Messungen wurden auf dem Testsystem, einer Alpha 21164 mit 433 MHz und 64 MB RAM durchgeführt. Bei allen Messungen liefen ausschließlich die an der Testumgebung beteiligten Threads.

5.1 Pagefault mit Nutzer-Pager (nicht Sigma 0)

Die erste Messung zeigt das Antwortzeitverhalten beim Seitenfehler und den Anteil der Kosten für die Funktionen der Mapping-Datenbank.

Die Messung wurde mit zwei Threads und Sigma 0 durchgeführt. Der Pager-Thread selbst hat Sigma 0 als zugeordneten Pager.

Zuerst wurde der Adreßraum linear von Sigma 0 an den Pager-Adreßraum gemappt. Danach wurde der Nutzer-Thread aktiviert, welcher linear auf seinen Adreßraum seitenweise lesend zugreift. Dabei wird jedesmal ein Seitenfehler generiert. Die untere Kurve zeigt das Zeitverhalten für die Datenbankoperationen, die obere Kurve die vollständige Interprozeßkommunikation. Die Nadeln aller 256 Takte entstehen durch den im Kern ausgelösten Seitenfehler, wenn die nächste freie Seite für Mappingeinträge eingeblendet wird.

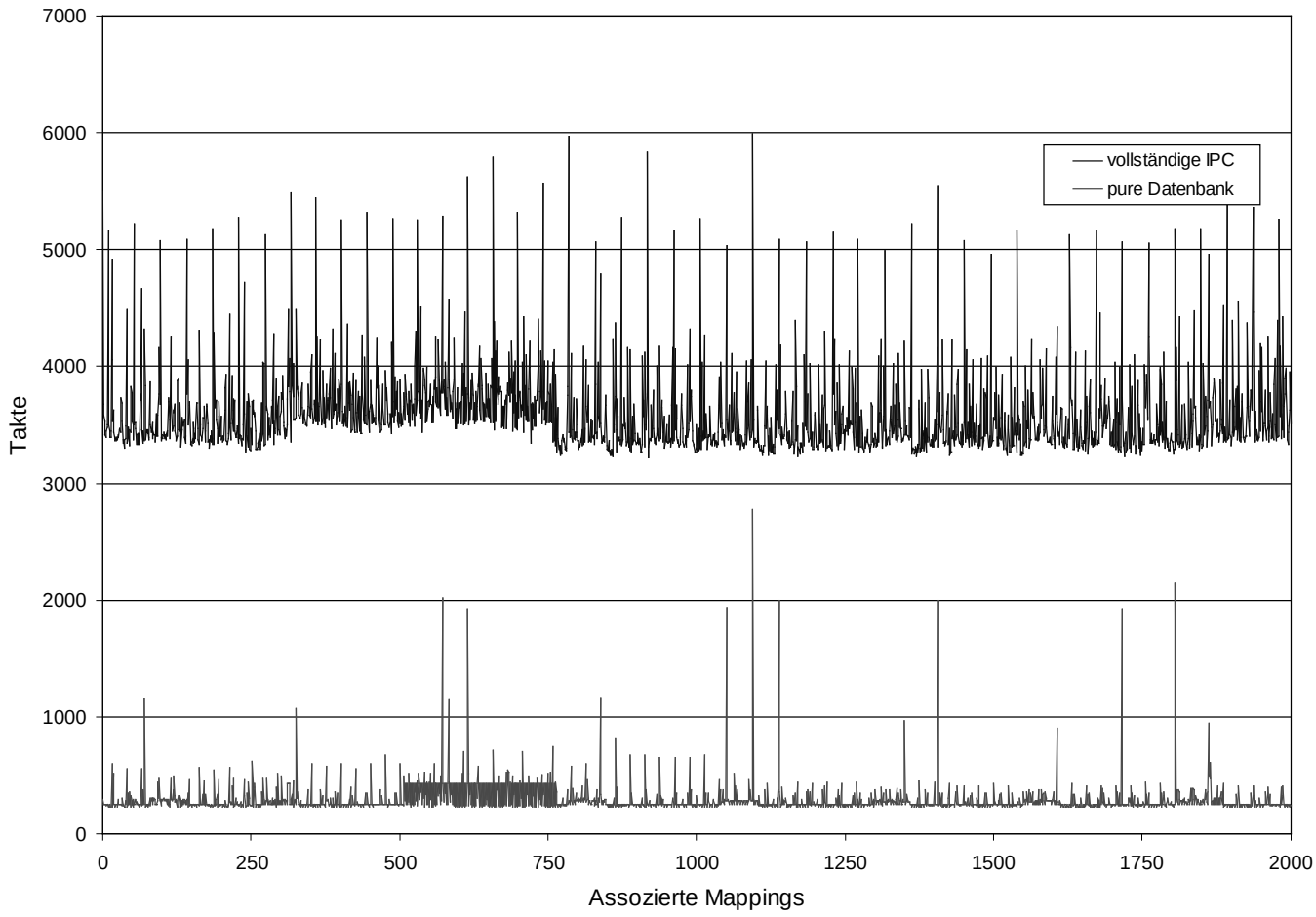

Abbildung 8: Seitenfehler mit Nutzertask als Pager (nicht Sigma 0)

5.2 Seitenfehler mit Sigma 0 als Pager

Die zweite Messung wurde mit nur einem Thread und Sigma 0 als Pager durchgeführt. Der Hauptunterschied besteht darin, daß Sigma 0 keine Seitentabelle besitzt und damit auch kein Quellmapping. Somit wird lediglich ein neuer Eintrag allokiert und mit den entsprechenden Werten gefüllt. Die reduzierte Operation spiegelt sich deutlich in der unteren Kurve wieder. Das vollständige Eintragen nimmt nur noch 60 Takte in Anspruch, im Gegensatz zu 300 Takten in der vorherigen Messung. Außerdem ist in Abbildung 9 deutlich die Dauer der Seitenfehlerbehandlung im Kern mit ca. 2000 Takten zu erkennen.

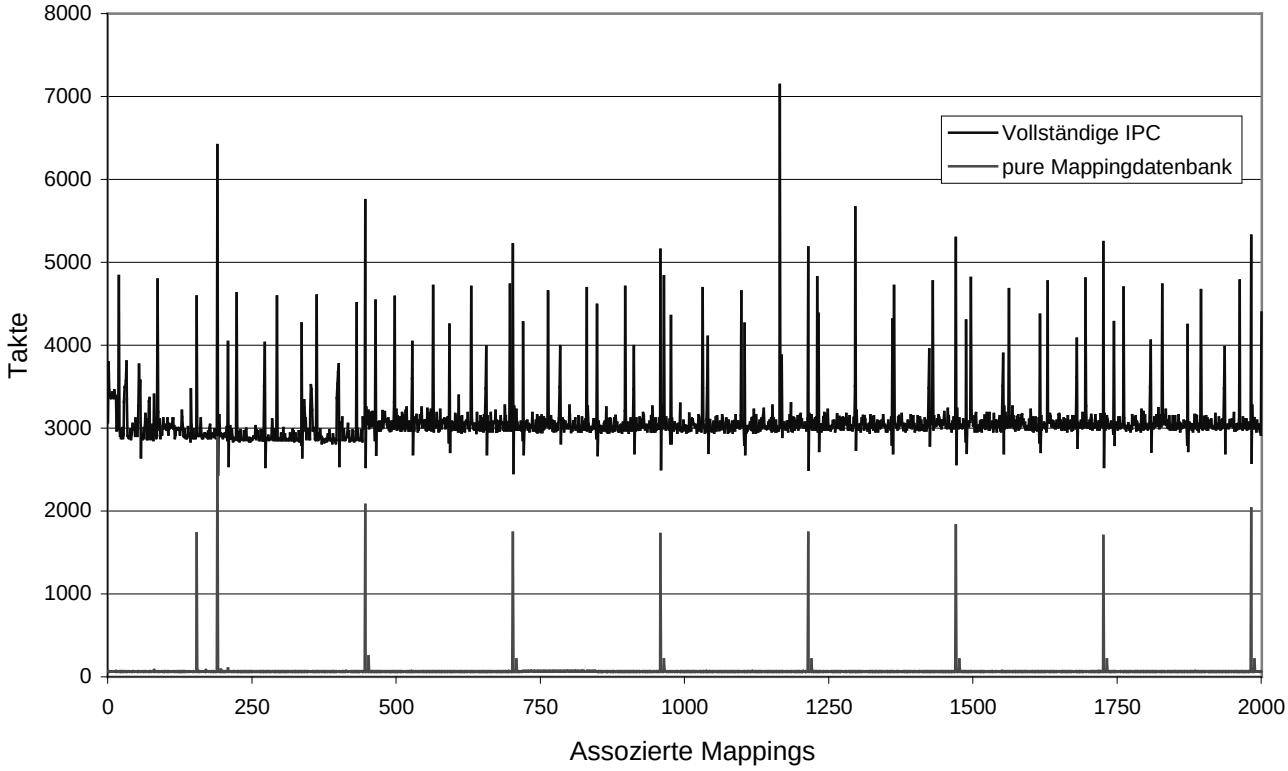

Abbildung 9: Seitenfehlerbehandlung mit Sigma 0 als Pager

5.3 Kosten der Flush-Operation pro Eintrag

Die Messung der Kosten für die Flush-Operation pro Mapping hat ergeben, daß die Zeiten für horizontales (ein Quellmapping - an viele Adreßräume) und vertikales Mappen (ein Adreßraum an den nächsten) fast identisch sind (Abbildung 11). Die etwas höheren Kosten für wenige Einträge sind damit zu erklären, daß am Anfang in den Code-Bereichen noch *Cache-Misses* auftreten, die bei mehr Einträgen vernachlässigt werden können.

Zur Messung wurden zwei Threads genutzt, die bei der horizontalen Messung eine Seite des Adreßraumes des ersten Threads mehrfach in den Adreßraum des zweiten Threads eingeblendet haben. Für die vertikale Messung wurde eine Seite abwechselnd von einem Adreßraum in den anderen eingeblendet (siehe Abbildung 10).

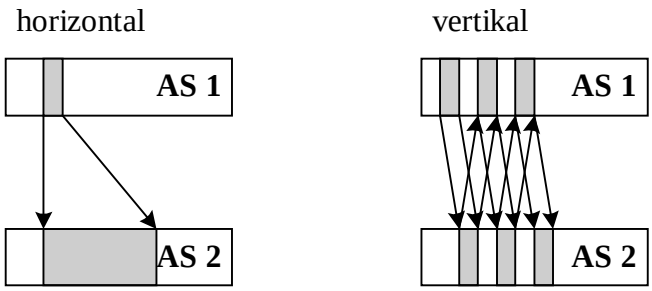

Abbildung 10: Assoziierte Einträge für Meßreihen der Flush-Operation

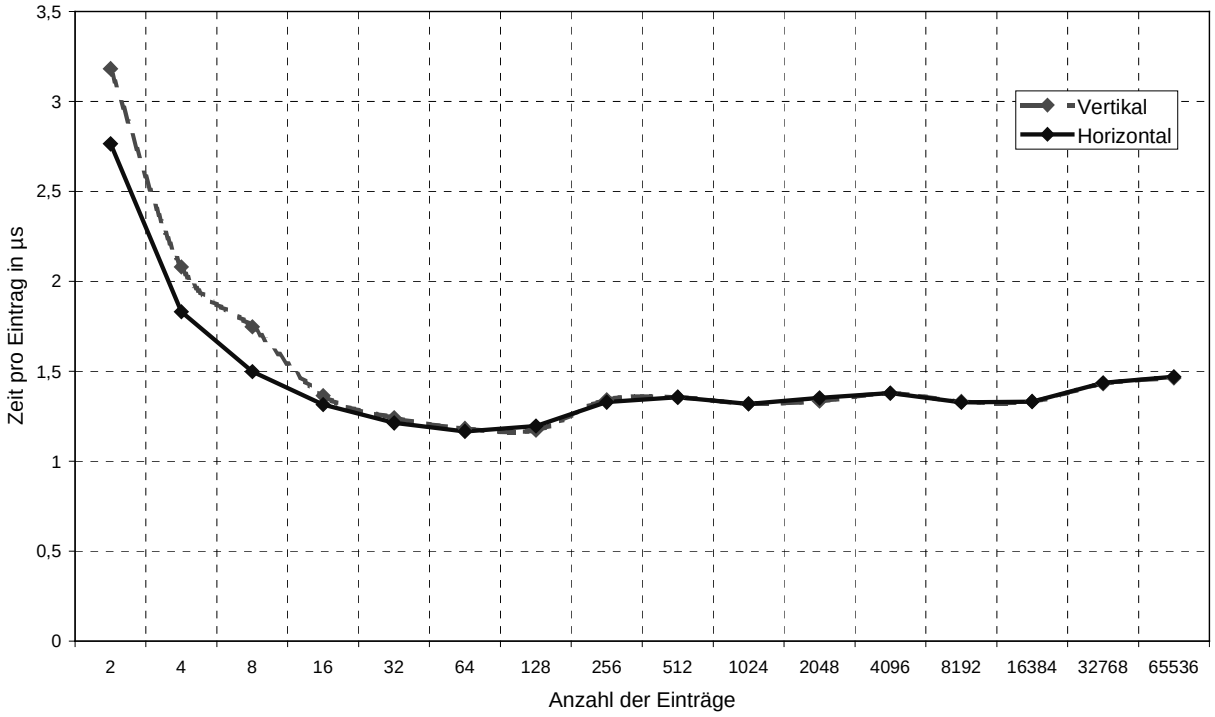

Abbildung 11: Kosten für die Flush-Operation pro Mapping.

5.4 Zusammenfassung der Meßergebnisse

Die Messungen haben gezeigt, daß die Mapping-Datenbank die Interprozeßkommunikation mit einem Anteil von 20% unwesentlich beeinflußt. Die eigentliche Datenbankmodifikation dauert im Durchschnitt 400 Takte, das entspricht ca. 1µs. Im ungünstigsten Fall müssen jedoch bis zu 3000 Takte, d. h. 7µs für diese Operation, angesetzt werden. Die vollständige Flexpage-Interprozeßkommunikation benötigt im Durchschnitt 3000 Takte und im schlechtesten Fall 7000, was ca. 16,5µs entspricht. Demzufolge sind auf der getesteten Plattform theoretisch über 60.000 Flexpage-Interprozeßkommunikationen pro Sekunde möglich.

Die Seitenfehlerbehandlung im Kern sollte noch optimiert werden, um die Spitzen von 2000 Takten bei der Map-Operation zu glätten.

Die Meßwerte der Flush-Operation sind für Echtzeitanwendungen gut geeignet, da die Dauer einfach abschätzbar ist. Im Normalfall können für das Entfernen eines Mappings maximal 3,5µs ansetzt werden.

Kapitel 6 - Schlußfolgerungen, Fragen und Ausblicke

Die realisierte Speicherverwaltung kann für Echtzeitsysteme Zusagen über Ressourcennutzung und Zeitverhalten mit relativ guten Randbedingungen geben. Dabei sind *Deadlock*-Situationen bereits durch den Systementwurf ausgeschlossen (siehe 3.4.1).

Als Erweiterung der derzeitigen Lösung wären die in 4.2 beschriebenen Ansätze für Mappingeinträge von Sigma 0 denkbar, wobei Aufwand und Nutzen auch im Hinblick auf die stärkere Speichernutzung kritisch gegenüber gestellt werden müssen.

Der Ansatz kann auch für andere Architekturen, wie zum Beispiel dem Intel-Prozessor, eingesetzt werden, wobei dort die in Hardware realisierten Seitentabellen keine so einfache Implementation zulassen. Für den Intel-Prozessor könnte aber beispielsweise zu jeder Seitentabelle eine zweite Seite allokiert werden, die die Referenz auf den zugehörigen Datenbankeintrag enthält.

Ein noch nicht behandelter Aspekt ist die Echtzeitfähigkeit der Flush-Operation. Zur Zeit kann vom Entwickler von Echtzeitapplikationen lediglich eine Abschätzung der Dauer einer Flush-Operation vorgenommen werden. Da aber die Anzahl der weitergegebenen Seiten nicht beschränkt ist, kann kein exaktes Zeitverhalten angenommen werden, außer wenn alle beteiligten Komponenten bekannt sind. Die Implementation einer zeitzusagefähigen Flush-Operation hängt davon ab, ob die Anzahl der erzeugten Einträge limitiert oder zumindest bekannt ist.

Die Datenbank kann durch verbessertes Cache-Verhalten und teilweise optimierten Code noch beschleunigt werden. Dies bedarf jedoch einer genaueren Analyse des Systems. Andererseits stellt sich die Frage, ob die Optimierung dieser relativ selten genutzten Funktionen den Aufwand lohnt.

Kapitel 7 - Zusammenfassung

In der vorliegenden Arbeit wurden Verfahren und Algorithmen für die hierarchische Speicherverwaltung für Echtzeitsysteme vorgestellt. Die Speicherdatenbank wurde für den L4 Kern auf DEC-Alpha, einem 64 Bit RISC-Prozessor, implementiert und ausgemessen.

Der L4 Kern arbeitet mit harten Prioritäten auf Thread-Ebene und nutzt die Interprozeßkommunikation zur Weitergabe von Zugriffsrechten auf den physischen RAM zwischen mehreren Adreßräumen.

Das Einfügen von Einträgen in die Datenbank wird atomar mit gesperrten Unterbrechungen durchgeführt; dies ermöglicht definitive Zeitzusagen für Echtzeitanwendungen. Das Löschen dagegen kann aufgrund der nicht bekannten Anzahl von betroffenen Einträgen nicht zeitdeterministisch realisiert werden. Deshalb wurde diese Operation fast vollständig unterbrechbar realisiert, um die Unterbrechungslatenz des L4 Kernes gering zu halten. Der Algorithmus zum Löschen der Seiten schließt im Entwurf Prioritätsinversion und Deadlock-Fälle durch Vererbung von Zeitscheiben vollständig aus.

Ein neuer Eintrag wird in die Datenbank innerhalb von 1µs bis maximal 7µs eingefügt. Das Entziehen von Zugriffsrechten auf eine Seite dauert pro assoziierter Seite maximal 3,5µs.

Danksagung

Ich danke den Mitarbeitern des Lehrstuhls Betriebssysteme der TU-Dresden für die Unterstützung bei der Erstellung dieser Arbeit, insbesondere Prof. Hermann Härtig und Sebastian Schönberg für die Betreuung des Beleges, Uwe Dannowski und Jean Wolter für Ratschläge und Hinweise.

Abbildungsverzeichnis

Literaturverzeichnis

[Lie95] Jochen Liedtke; On micro-kernel construction; In 15th ACM Symposium on Operating System Principles (SOSP), Seiten 237 250.

[SU98] Sebastian Schönberg, Volkmar Uhlig; Microkernel Memory Managment; Dresden University of Technology; May 1998

[Sch96] Sebastian Schönberg; L4 on Alpha, design and implementation. Technical Report CS-TR-407; Unviersity of Cambridge; 1996.

[HWL96] H. Härtig, J. Wolter und J. Liedtke, Flexible-sized Page Objects; In 5th International Workshop on Object Orientation in Operating Systems (IWOOOS) , Seiten 102 bis 106; Seattle, WA, Oktober 1996.

[Lie96] L4-Reference Manual, 486, Pentium and Pentium Pro, Version 2.0, GMD Arbeitspapier 1021 and Research Report RC 20549 IBM T. J. Watson Research Center, Yorktown Heights, September 1996