Konzepte von Betriebssystem-Komponenten

06.02.2003

Distributed Shared Memory OS

PLURIX

Axel Schlicker

Axel.J.Schlicker@informatik.stud.uni-erlangen.de

1. Einleitung

Plurix ist ein echt verteiltes System mit verteiltem Speicher. Es wurde an der Universität Ulm am Lehrstuhl für verteilte Systeme unter der Leitung von Prof. Dr. Peter Schulthess entwickelt.

Plurix wurde komplett in der Programmiersprache Java geschrieben, also auch der Kernel und die Gerätetreiber. Für die JavaCode-Übersetzung wurde dadurch die Entwicklung einer neuen Java Virtual Mashine (JVM) nötig.

Ziel des Forschungsprojektes war es, die laut den Entwicklern einzigartige Fusion der Prinzipien Verteiltes System, datentypsichere Sprache, DSM und Optimistische Synchronisation auf deren Tauglichkeit bzw. Funktionalität zu prüfen.

2. Grundlagen

1 Prinzip echt verteilter Systeme

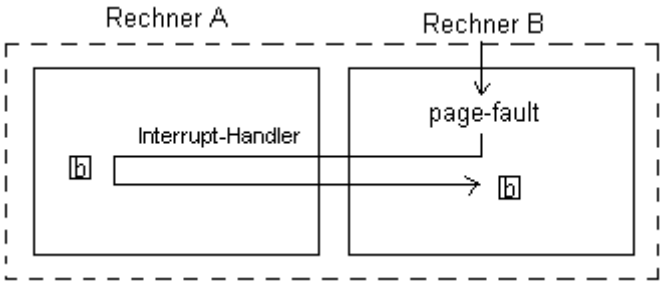
Im Gegensatz zu ‚normal' verteilten Betriebssystemen, bei denen alle Rechner an einem zentralen Server angebunden sind, entfällt bei ‚echt' verteilten Betriebssystemen diese zentrale Komponente. Es gilt hier der Grundsatz, dass alle Rechner gleich berechtigt sein und sich dem Anwender als eine große

Einheit präsentieren sollten, welcher durch das Fehlen eines Zentrums unterstützt wird. Ein weiterer Vorteil, der sich daraus ergibt, ist die Tatsache, dass nun beim Ausfall nur einer Komponente, das Restsystem weiterarbeiten kann, was bei Ausfall des Servers nicht möglich wäre. In der Theorie sollte es möglich sein, beliebig viele Endgeräte anzuschliessen, was in der Praxis jedoch durch starke Synchronisationsprobleme eingeschränkt wird.

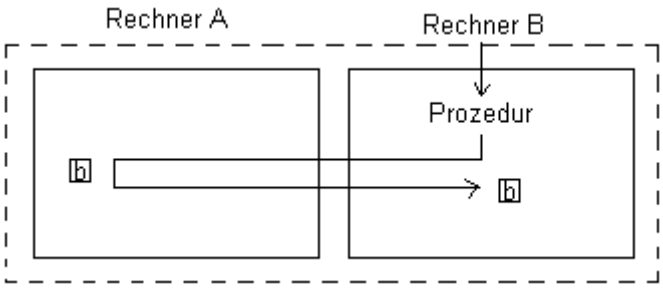
2.2 DSM – Distributed Shared Memory

Bei dem Prinzip des verteilten gemeinsamen Speichers wird dem Anwendungsprogramm die Illusion vermittelt, es existiere ein großer gemeinsamer Speicher, der sich über alle Knoten des Systems erstreckt. Dabei können zwei Varianten des DSM unterschieden werden:

a) <u>betriebssystembasiertes DSM</u>
Ähnlich wie bei virtuellem Speicher wird hier beim Zugriff auf eine nicht lokal im Speicher vorhandene Speicherseite der Zugriff mittels einer Speicherverwaltungseinheit (MMU) abgefangen, und der generierte page-fault dem Betriebssystem mitgeteilt. Nun wird die gewünschte Seite durch Ausführung eines Interrupt-Handler im Kernel von einem anderen Knoten geholt. (Zugriffsgranularität 4-8 kB Seiten)



b) <u>objektbasiertes DSM</u>
Hier erfolgt der Zugriff auf Objekte nie direkt, sondern immer durch Zugriffsprozeduren. Diese sind leicht so modifizierbar, dass beim Start einer solchen Prozedur vor dem Zugriff geprüft wird, ob die gewünschte Seite lokal vorhanden ist. Ist die nicht der Fall, wird die Seite von anderen Knoten geholt. Dies sichert eine enorme Flexibilität und Granularität, stellt jedoch auch ein Problem dar, insofern, dass bei jedem Zugriff die Lokalitätsprüfung abläuft.



3. Architekturkonzept von Plurix

Das in Plurix verwendete Prinzip des betriebsystembasierten DSM wird durch einen verteilten Heap (DHS – Distributed Heap Storage) verwirklicht.

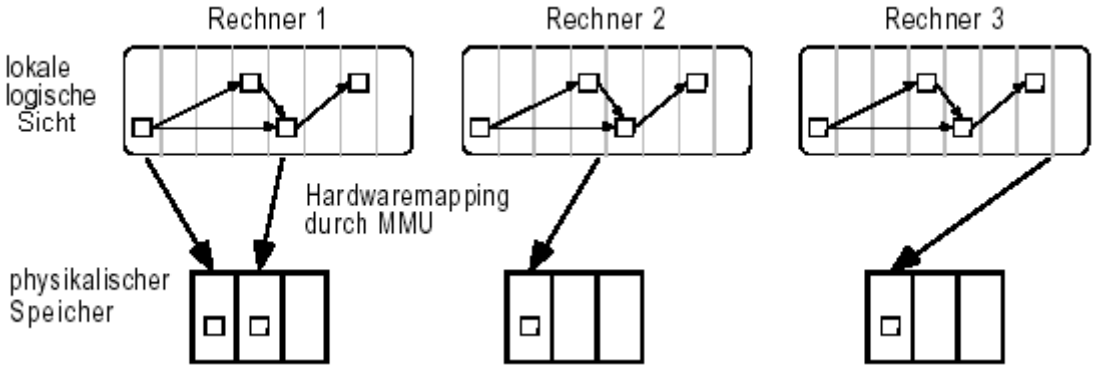
Verteilter Heap (DHS):

In Plurix gibt es einen großen Adressraum, der sich über alle Knoten erstreckt und der von jedem Knoten aus, zu jeder Zeit einsehbar ist.
Da die Daten auf unterschiedlichen Rechner verteilt liegen, muss das Betriebsystem dafür sorgen, dass sich die Daten zur rechten Zeit am rechten Ort befinden.

Jeder Rechner besitzt eine eigene MMU, die Teile des logischen Adressraums auf den physikalischen Speicher des anfragenden Rechners abbildet. Das Betriebsystem hat hierbei die Eindeutigkeit der Abbildung sicherzustellen, um unerwünschte Nebeneffekte zu verhindern.

Datenobjekte werden im globalen Adressraum prinzipiell auf gleiche Weise angelegt und verwaltet wie in lokalen dynamischen Heaps. Auch Code und Stackbereiche werden als spezielle Heapobjekte angesehen und auf dem globalen Heap abgelegt.

Der Zugriff auf beliebige Heapobjekte erfolgt über Derefferenzierung von typgebundenen Zeigern. Dabei muss der physikalische Speicherort der Daten nicht bekannt sein, es reicht auch der Ort im globalen Adressraum. Soll auf ein Objekt zugegriffen werden, wird also ein Zeiger auf das Objekt geholt und dann diese Speicherseite durch Derefferenzierung in den physikalischen Speicher abgebildet, falls sie nicht schon lokal vorhanden war.
Dadurch, dass mit Java in Plurix eine typensichere Sprache verwendet wurde, sind ungültige Zeigerwerte (C) nicht möglich!



Durch die globale Sicht eines jeden Knoten, entstehen aber auch einige Synchronisationsprobleme. In Plurix wurde dieses Problem mittels verteilten Transaktionen gelöst. Da dies aber ein sehr komplexes Thema darstellt, wird das Prinzip etwas vereinfacht dargestellt.

Synchronisation - Verteilte Transaktionen

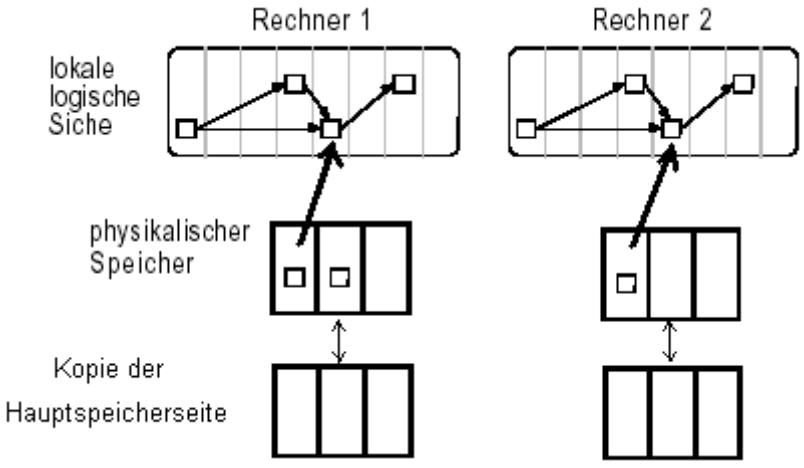
Das aus anderen Betriebssystemen bekannte Synchronisationsverfahren mit Semaphore und Monitor, bei dem kritische Abschnitte definiert werden, in denen gleichzeitig nur ein Prozess vorhanden sein darf und alle anderen blockiert werden (pessimistisches Verfahren), wurde in Plurix durch das System von verteilten Transaktionen ersetzt.

Der Ablauf einer Benutzersitzung wird in einfache Transaktionen unterteilt (Mausklick, Tastendruck, usw.). Dabei wird zu Beginn einer jeden Transaktion der Zugriff auf alle Speicherseiten durch das Betriebsystem gesperrt, und anschließend eine Kopie der Hauptspeicherseite angelegt.

Es wird nun davon ausgegangen, dass keinerlei Konflikte zwischen den Zugriffen der einzelnen Transaktionen der Benutzer auftreten (optimistisches Verfahren), und deshalb wird allen Transaktionen der Zugriff auf alle Speicherseiten gewährt, obwohl diese gesperrt sind. Zwangsläufig fängt die MMU jeden Zugriff einer Transaktion ab und generiert einen page-fault, der vermerkt wird.

Diese Vermerke werden nun auf Kollisionen geprüft. Tritt eine solche auf, werden alle Transaktionen bis auf eine, nämlich die erste (FCFS), zurückgesetzt. Dieses Rücksetzen geschieht durch das Rückkopieren der vorhin gespeicherten Hauptspeicherseite, was den Originalzustand wieder herstellt.

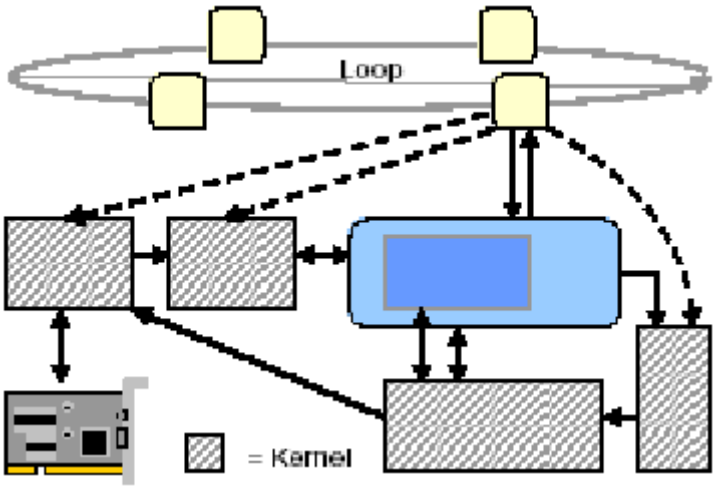
Anschließend gibt das Betriebsystem den Zugriff auf die Speicherseite frei, wodurch die Transaktion beendet werden kann.



Plurix – Kernel

Der Plurix-Kernel besteht aus einer MMU, unterschiedlichen Geräte-Treibern, Methoden für Exceptions und Interrupts und einigen weiteren Komponenten für spezielle Verfahren. Der Kernel besitzt einen eigenen festgelegten Speicherbereich, der nicht mit dem DHS zusammenhängt.

Hätte der Kernel keinen separaten Adressraum, würden dadurch einige Probleme entstehen. Zum Beispiel könnten dann alle User, die ja globale Sicht auf dem DHS haben, auch die Komponenten des Kerns sehen, und dadurch auch auf diese zugreifen.
Dadurch wären dann wieder Schutzmechanismen von Nöten, die man sich durch den geteilten Adressraum nun sparen konnte.



4. Besonderheiten von Plurix

* für Plurix wurde eigens eine JVM erstellt, die den Java-Code direkt in Maschinen-Code übersetzt
* Plurix besteht aus 3100 – 20000 Zeilen Code
* Plurix lässt sich komplett von einer Floppy booten

5. Literaturverzeichnis

[1] www.plurix.de
[2] www-vs.informatik.uni-ulm.de
[3] www-vs.informatik.uni-ulm.de/projekte/Plurix/DissertationTraub.pdf i.W. Kapitel 1-3