JX: Eine adaptierbare Java-Betriebssystemarchitektur

Michael Golm, Jürgen Kleinöder
Universität Erlangen-Nürnberg
Lehrstuhl für Betriebssysteme
Martensstr. 1, D-91058 Erlangen
{golm, kleinoeder}@informatik.uni-erlangen.de

1 Einleitung

Die Mehrzahl heutiger Betriebssysteme basiert auf einer hardwareunterstützten Aufteilung des Systems in eine privilegierte und eine nichtprivilegierte Ausführungsumgebung. Diese Zweiteilung macht es sehr schwierig, die Funktionalität des privilegierten Bereiches - des Kernels - zu erweitern oder anzupassen. Weiterhin bestehen viele, oft implizite, Abhängigkeiten zwischen den Komponenten des Kernels. Bestimmter Code kann nur in einem bestimmten Kontext ausgeführt werden, z.B. Interrupt- oder Prozeßkontext. Die Abhängigkeiten zwischen den Komponenten und die Abhängigkeit einer Komponente von bestimmten Eigenschaften der Ausführungsumgebung kann in den seltensten Fällen automatisch erkannt werden. Durch die damit erzeugte Komplexität kann eine Komponente nur von einem erfahrenen Programmierer modifiziert werden.
Die JX Betriebssystemarchitektur versucht, die beiden beschriebenen Probleme zu lösen: schlechte Adaptierbarkeit des privilegierten Bereichs und Komplexität der Komponenten.

2 Die JX-Systemarchitektur

Im Gegensatz zum addressraum-basierten Schutz traditioneller Betriebssysteme basiert die Sicherheit in JX auf der Verwendung einer sicheren Sprache - des Java Bytecodes [Gos95].
Die Idee der software-basierten Sicherheit ist nicht neu. Das SPIN-Betriebssystem [PaBe96] benutzt Modula-3 für Erweiterungsmodule, sogenannte Spindles. VINO [Selzer96] verwendet Software-Fault-Isolation (SFI) [Wahbe96], um Erweiterungsmodule in einer Sandbox ablaufen zu lassen und Transaktionen, um Ressourcenmißbrauch zu unterbinden. Jedes dieser Systeme versucht, ein traditionelles Betriebssystem durch die Verwendung sicherer Sprachen oder SFI zu erweitern. Im JX-System liegen alle Betriebssystemkomponenten, wie z.B. Dateisystem, Netzwerkstack und Gerätetreiber, in Form von Java-Bytecode vor. Der statische Betriebssystemkern ist minimal, enthält allerdings eine Java-Laufzeitumgebung [LiYe96]. Die Komponenten werden von einem Translator [Kopp98] von Bytecode in Maschinencode übersetzt.

2.1 Isolation und Kooperation

Im Unterschied zum monolithischen JavaOS [JavaOS] verwendet JX *Domains*, um verschiedene Ausführungsumgebungen voneinander zu isolieren. Jede Domain besitzt ihren eigenen Heap und eigene Threads. Domains kommunizieren ausschließlich über *Portale*. Ein Portal ist eine Objektreferenz, die vom System speziell behandelt wird, aber wie eine normale Objektreferenz verwendet wird. Diese Objektreferenz wird über ein Java-Interface verwendet. Damit sind die Heaps der einzelnen Domains bis auf die Portale vollständig entkoppelt und können jeweils einen eigenen Garbage Collector besitzen, welcher unabhängig von anderen Domains arbeitet. Domains können unterschiedliche GC-Strategien und Implementierungen verwenden, z.B. kopierende oder nicht-kopierende Kollektoren.
Beim Aufruf über ein Portal erfolgt eine Threadumschaltung auf einen dem Portal zugeordneten Thread in der Zieldomain. Damit sind auch die Threads und die Scheduler der einzelnen Domains unabhängig voneinander. Während in einer Domain ein zeitscheibenbasierter Scheduler arbeitet, können die Threads einer anderen Domain nur an dedizierten Punkten verdrängt wer-

den. Das Scheduling zwischen den Domains erfolgt vollständig preemtiv, d.h. eine Domain kann jederzeit angehalten werden.

Es existiert eine spezielle Domain - die *Domain-Zero*- welche Zugriff auf die Hardware des Rechners ermöglicht und als einzige Domain in C bzw. Assembler implementiert ist. Die DomainZero exportiert Portale, um ihre Dienste anderen Domains zur Verfügung zu stellen.

Die Interrupt-Behandlung erfolgt ebenfalls über Portale. Eine Domain kann ein Portal eines speziellen Typs bei der DomainZero als Interrupthandler registrieren. Tritt nun ein Interrupt auf, wird auf den Portal-Handler-Thread umgeschaltet.

2.2 Arten von Komponenten

Es existieren vier Arten von Betriebssystemkomponenten: Interfacekomponenten, Dienstkomponenten, Domainkomponenten und Bibliotheken. Jede dieser Betriebssystemkomponente besteht aus einer Menge von Klassen oder Interfaces. Eine *Dienstkomponente* stellt einen Dienst zur Verfügung, z.B. die Ansteuerung eines Ethernetcontrollers. Dieser Dienst ist über Portale ansprechbar, deren Interfaces in der entsprechenden *Interfacekomponente* definiert sind. Mehrere Dienstkomponenten können zu einer Domain zusammengestellt werden. Eine Domain enthält genau eine *Domainkomponente*, welche die Dienstkomponenten “verdrahtet”. Eine *Bibliothek* enthält wiederverwendbare Klassen, z.B. die Klassen der JDK-Klassenbibliothek.

Der Code einer Komponente ist im System nur einmal vorhanden, auch wenn die Komponente in verschiedene Domains geladen wird.

2.3 Externe Ressourcenverwaltung

Die von einer Domain benötigten Ressourcen, z.B. Heap und Threads, können von einer anderen Domain verwaltet werden. Der Garbage Collector und der Scheduler arbeiten dann in einer anderen Domain und werden vom Laufzeitsystem der verwalteten Domain über Portale angesprochen.

3 JX vs. metaXa

Der Vorläufer des JX-Systems, metaXa [Golm97], konnte für jedes Objekt eine spezielle Methodenaufruf- oder Locking-Strategie definieren. Wir erkannten allerdings, daß diese feine Granularität nicht nur Vorteile besitzt, sondern ein gewaltiges Konfigurationsproblem darstellt und die Konfiguration abhängig wird von den internen Details einer Implementierung.

4 Status

JX läuft auf einem Pentium-PC und unterstützt ISA und PCI-Bus. Es existieren Komponenten zur Ansteuerung eines IDE-Controllers, des Ethernetcontrollers 3C905B von 3Com, der Matrox-Graphikkarte G200, sowie von Framegrabberkarten mit Bt484 Chip. Weiterhin existiert eine Filesystemkomponente [Weiss00] (Blocklayout kompatibel zu ext2). Ein TCP/IP-Protokollstack und die Anbindung an AWT befindet sich in Entwicklung.

5 Literatur

Golm97 M. Golm, *Design and Implementation of a Meta Architecture for Java*. Diplomarbeit , Universität Erlangen, IMMD 4, Jan. 1997.

Gos95 J. Gosling, Java Intermediate Bytecodes, *Workshop on Intermediate Representations*, ACM Sigplan Notices 30(3), March 1995.

JavaOS *JavaOS for Business Reference Manual*, Version 2.1 (http://www.sun.com/javaos/business/).

Kopp98 H. Kopp, *Design und Implementierung eines maschinenunabhängigen Just-in-Time-Compilers für Java*, Diplomarbeit (masters thesis), Universität Erlangen, IMMD 4, Oct. 1998.

LiYe96 T. Lindholm, F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley 1996.

PaBe96 Przemyslaw Pardyak and Brian Bershad. Dynamic binding for an extensible system. In *2nd OSDI*, pp. 201-212, Seattle, WA, 1996.

Selzer96 M. I. Seltzer, Y. Endo, C. Small, and K. A. Smith. Dealing With Disaster: Surviving Misbehaved Kernel Extensions. In *2nd OSDI*, pages 213--227, 1996.

Wahbe96 R. Wahbe, S. Lucco, T. Anderson, S. Graham. Efficient Software-based fault isolation. In *Proc. 14th SOSP*, June 1993, pp. 203-216.

Weiss00 A. Weissel, *Ein offenes Dateisystem mit Festplattensteuerung für metaXaOS*. Studienarbeit , Universität Erlangen, IMMD 4, Apr. 2000.