

The Structure of a Type-Safe Operating System

Der Technischen Fakultät der
Universität Erlangen-Nürnberg
zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Michael Golm

Erlangen - 2002

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	16. 09. 2002
Tag der Promotion:	17. 12. 2002
Dekan:	Prof. Dr. rer. nat. A. Winnacker
Berichterstatter:	Prof. Dr. rer. nat. F. Hofmann

Michael Golm

The Structure of a Type-Safe Operating System

The architecture of traditional operating systems relies on address-based memory protection. To achieve flexibility at a low cost operating system research has recently started to explore alternative protection mechanisms, such as type safety. This dissertation presents an operating system architecture that completely replaces address-based protection with type-based protection. Replacing such an essential part of the system leads to a novel operating system architecture with improved robustness, reusability, configurability, scalability, and security.

The dissertation describes not only the design of such a system but also its prototype implementation and the performance of initial applications, such as a file system, a web server, a data base management system, and a network file server. The prototype, which is called JX, uses Java bytecode as its type-safe instruction set and is able to run existing Java programs without modifications.

The system is based on a modular microkernel, which is the only part of the system that is written in an unsafe language. Light-weight protection domains replace the heavy-weight process concept of traditional systems. These domains are the unit of protection, resource management, and termination. Code is organized in components that are loaded into domains. The portal mechanism—a fast inter-domain communication mechanism—allows mutually distrusting domains to cooperate in a secure way.

This dissertation shows that it is possible to build a complete and efficient general-purpose time-sharing operating system based on type safety.

Part of the material presented in this dissertation has previously been published in the following conference proceedings:

Michael Golm, Meik Felser, Christian Wawersich, and Jürgen Kleinöder. **The JX Operating System**. In *Proc. of the USENIX Annual Technical Conference*, Monterey, CA, USENIX Association, pp. 45-58, June 10-15, 2002.

Michael Golm, Jürgen Kleinöder, and Frank Bellosa: **Beyond Address Spaces - Flexibility, Performance, Protection, and Resource Management in the Type-Safe JX Operating System**. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloß Elmau, Germany, IEEE Computer Society Press, pp. 1-6, May 20-23, 2001.

Meik Felser, Christian Wawersich, Michael Golm, and Jürgen Kleinöder. **Execution Time Limitation of Interrupt Handlers in a Java Operating System**. In *Proc. of the 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, Sept. 2002.

Michael Golm and Jürgen Kleinöder. **JX: Eine adaptierbare Java-Betriebssystemarchitektur**. *GI-Herbsttagung: Mobile Computing*, Munich, Germany, November 16-17, 2000.

Contents

1. Introduction	1
1 Motivation	1
1.1 Robustness, security, and reliability	2
1.2 Configurability, reusability, and dedicated systems	2
1.3 Extensibility and fine-grained protection	3
2 Design objectives	3
3 Contributions.....	4
4 Dissertation outline	4
2. Background.....	5
1 Operating system architectures	5
1.1 Monolithic systems	5
1.2 Microkernels	6
1.3 Single address-space systems	7
2 Protection mechanisms.....	7
2.1 Address-space based protection	8
2.2 Software Fault Isolation	8
2.3 Proof-carrying code	8
2.4 Virtual machines	8
2.5 Type-safe instruction set	9
3 Resource protection and management mechanisms.....	12
3.1 Hierarchical resource management	12
3.2 Resource containers	12
3.3 Lightweight transactions	12
3.4 Path-based resource management	12
3.5 User-defined resource management	13
4 Modularity and reusability	13
5 Extensibility and openness.....	14
6 Summary.....	15
3. Architectural Overview	17
1 Modular microkernel.....	18

2	Protection domains.....	19
3	Portals and services.....	20
4	Reusable system components.....	21
5	Bytecode-to-nativecode translator.....	22
6	Fast portals.....	23
7	Bootstrapping.....	23
8	Performance measurements.....	23
9	Summary.....	25

4.	Domains.....	27
1	Structure of a domain.....	27
2	Lifecycle of a domain.....	27
	2.1 Creation	27
	2.2 Termination	28
3	Isolation.....	29
	3.1 Resource management	29
	3.2 Execution environment	30
4	Domain ID.....	31
5	Performance evaluation.....	31
6	Summary.....	34

5.	Portals.....	35
1	Design principles.....	35
2	Programming model.....	35
3	Parameter passing.....	37
	3.1 Copying algorithm	38
	3.2 Passing special objects	39
	3.3 Implicit portal parameters	42
	3.4 Summary	43
4	Garbage collection and portal invocations.....	43
5	Life-cycle management of services.....	44
	5.1 Service creation	44
	5.2 Service termination	44
6	Implementation.....	45
	6.1 Thread state transitions during a portal call	46
	6.2 Premature call termination	49
	6.3 The translator and portal communication	49
	6.4 Fast portals	50

7	Performance evaluation	51
7.1	Null portal invocation	51
7.2	Portal invocation path	52
7.3	Parameter copying cost	52
8	Summary	56
6.	Components.....	59
1	Component types	59
2	Shared components	60
3	Code reuse between colocated/dislocated configuration	62
4	Component relationships	62
4.1	Dependencies	62
4.2	Compatibility	64
5	Portal communication and components	64
6	Translation to machine code.....	66
6.1	Method table creation and interfaces	68
6.2	Optimizations	68
6.3	Java-to-native control transfer	70
7	Summary.....	71
7.	Processor Scheduling	73
1	Microkernel scheduling support	73
2	Scheduler configurations	76
2.1	Global kernel scheduler	76
2.2	Global kernel schedulers and domain-local kernel schedulers	76
2.3	Global kernel scheduler and domain-local Java schedulers	78
3	Interaction between domains and the scheduler	79
4	Concurrency control	80
4.1	Kernel-level locking	80
4.2	Domain-level locking	80
4.3	Inter-domain locking	80
5	Multiprocessor support.....	81
6	Performance evaluation	81
7	Summary.....	82
8.	Memory Management	83
1	Global and domain-local memory management	83
2	Heap	84

2.1	GC interface	86
2.2	Moving special objects	86
2.3	Garbage collection and interrupt handling	89
2.4	Garbage collecting and timeslicing	89
2.5	Garbage collection of portals and memory objects	90
2.6	Future work	90
2.7	Summary	91
3	Memory Objects.....	91
3.1	Lifecycle of memory objects	91
3.1.1	Creation	91
3.1.2	Destruction	91
3.2	Implementation	92
3.3	Problems	94
3.4	Buffer management	94
4	Performance evaluation	96
4.1	Global memory management	96
4.2	Stack size check	96
4.3	Memory objects	96
4.3.1	Delayed MCB creation	98
4.3.2	MappedMemory	98
5	Summary.....	99

9. Device Drivers 101

1	Device-driver architecture of traditional systems.....	101
2	JavaOS Driver Architecture	102
3	JX Driver Architecture	102
3.1	Interrupt handling	103
3.2	Register access	104
3.3	Timer	104
3.4	Garbage collection	105
4	Robustness.....	106
5	Device driver framework.....	107
5.1	Device identification	108
5.2	Example classes of device drivers	109
5.2.1	The network driver	109
5.2.2	The disk driver	109
5.2.3	The framebuffer driver	110
6	Performance evaluation	110
7	Lessons learned	112
8	Summary.....	112

10. Security Architecture	115
1 Security problems in traditional systems	115
1.1 Operating systems <i>115</i>	
1.2 Java Security <i>116</i>	
2 Requirements for a security architecture	117
2.1 Saltzer & Schroeder <i>117</i>	
2.2 Confinement <i>117</i>	
2.3 Integrity <i>118</i>	
2.4 Separation of policy and enforcement <i>118</i>	
2.5 Suitable programming language <i>118</i>	
2.6 Performance <i>118</i>	
3 Capabilities	119
4 JX as a capability system	119
5 The reference monitor	120
5.1 Making an access decision <i>121</i>	
5.2 Controlling portal propagation <i>122</i>	
6 Principals.....	124
7 Revocation.....	124
7.1 Revocation of portals <i>124</i>	
7.2 Revocation of fast portals and memory objects <i>125</i>	
8 Structure of the Trusted Computing Base.....	125
9 Device drivers	127
10 Tamper-resistant auditing.....	128
11 Trusted path	128
12 JDK - System classes and protection domains	129
13 Maintaining security in a dynamic system	130
14 Securing servers	130
14.1 Securing the file server <i>131</i>	
14.2 Securing the database server <i>132</i>	
15 Evaluation.....	132
16 Summary.....	135

11. Related Work	137
1 Architectural aspects.....	139
1.1 Isolation <i>139</i>	
1.2 Communication <i>140</i>	
1.3 Sharing <i>141</i>	
1.4 Resource accounting <i>141</i>	
1.5 User-defined resource management <i>141</i>	

1.6 Scalability	141
1.7 Security	142
1.8 Reusability	142
2 Summary	143

12. Evaluation 145

1 Functional Evaluation	145
1.1 Filesystem	145
1.2 Network File System	146
1.3 Window manager	146
1.4 Database Management System	148
1.5 Webserver	148
2 Performance evaluation	148
2.1 Filesystem	148
2.2 Network File System	150
2.3 Database Management System	151
2.4 Web server	151
3 Detailed performance analysis of the file system	152
3.1 Domain structure	153
3.2 Translator configuration	153
3.2.1 Inlining	153
3.2.2 Inlining of fast portals	153
3.2.3 Safety checks	155
3.3 Memory revocation	155
3.4 Cost of the open scheduling framework	156
3.5 Security	156
3.6 Summary: Fastest safe configuration	156
4 Detailed performance analysis of the NFS server	157
4.1 Method timing	157
4.2 Event tracing	159
4.3 Memory	159
5 Summary	161

13. Conclusions 169

1 Contribution	169
2 Meeting the objectives	169
3 Design summary	170
4 Future work	171
5 Final remarks	171

List of Figures

Chapter 1: Introduction

Chapter 2: Background

Figure 2.1:	Type-based memory protection	10
Figure 2.2:	Relation between Java source code, Java bytecode, and machine code	11

Chapter 3: Architectural Overview

Figure 3.1:	Structure of the JX system	17
Figure 3.2:	Structure of the JX microkernel	19
Figure 3.3:	Portal invocation	20
Figure 3.4:	Two different system configurations using the same set of components	22

Chapter 4: Domains

Figure 4.1:	Structure of a domain	27
Figure 4.2:	Naming portal	28
Figure 4.3:	Protection domain memory footprint	33
Figure 4.4:	Break down of the memory footprint of a JX domain	33

Chapter 5: Portals

Figure 5.1:	The interface of a portal	36
Figure 5.2:	Portal client code	36
Figure 5.3:	Service implementation	36
Figure 5.4:	Portal to service in other domain	40
Figure 5.5:	Portal to service in target domain	41
Figure 5.6:	Pruning a large object graph at service objects	42
Figure 5.7:	Problematic connections	43
Figure 5.8:	Portal data structures	46
Figure 5.9:	Send operation	47
Figure 5.10:	Receive operation	48

Figure 5.11:	States of a client thread during a portal call	49
Figure 5.12:	Fast portals	51
Figure 5.13:	Events logged during the send operation	53
Figure 5.14:	Events logged during the receive operation	54
Figure 5.15:	Event sequences during a loop of portal invocations.	55
Figure 5.16:	Portal parameter copying cost	57
Figure 5.17:	Small number of object parameters	57

Chapter 6: Components

Figure 6.1:	Notation for system configurations	60
Figure 6.2:	Data structures for the management of components	61
Figure 6.3:	Component compatibility and dependability	63
Figure 6.4:	Component compatibility and portal communication	65
Figure 6.5:	Path through components	67
Figure 6.6:	Inlining of fast portal methods	69

Chapter 7: Processor Scheduling

Figure 7.1:	Scheduler configurations	73
Figure 7.2:	Thread states	74
Figure 7.3:	Interface between the VM and the global scheduler	75
Figure 7.4:	Interface between global scheduler and domain-local scheduler	76
Figure 7.5:	Example interaction between global scheduler and local scheduler	77
Figure 7.6:	ThreadManager and CPUState interface	79
Figure 7.7:	Multiprocessor scheduling in JX	82

Chapter 8: Memory Management

Figure 8.1:	Copying garbage collection	85
Figure 8.2:	Garbage collector interface	86
Figure 8.3:	Management of Thread Control Blocks during a portal communication	88
Figure 8.4:	Relation between TCBs, inter-domain TCB references, and stacks	88
Figure 8.5:	GC in kernel functions	90
Figure 8.6:	Data structures for the management of memory objects	92
Figure 8.7:	Mapping a memory range to a class	93
Figure 8.8:	Principle of buffer management	95
Figure 8.9:	Time between events during the creation of a memory object.	97
Figure 8.10:	Measured code sequence for JX MappedMemory access	99
Figure 8.11:	Measured code sequence for C struct access	99

Chapter 9: Device Drivers

Figure 9.1:	Example memory layout and structure of a JX device driver	103
Figure 9.2:	InterruptManager and InterruptHandler interfaces	104

Figure 9.3:	TimerManager interface	105
Figure 9.4:	AtomicVariable interface	107
Figure 9.5:	Split device driver	108
Figure 9.6:	Device and DeviceFinder interface	108
Figure 9.7:	Network device interface	109
Figure 9.8:	BlockIO device interface	110
Figure 9.9:	Framebuffer device interface	110
Figure 9.10:	Interrupt latency in JX	111
Figure 9.11:	Measured interrupt response time	111

Chapter 10: Security Architecture

Figure 10.1:	Reference monitor interface	123
Figure 10.2:	Information interfaces	123
Figure 10.3:	Typical TCB structure	126
Figure 10.4:	Device driver information flow	127
Figure 10.5:	Filesystem layers	131

Chapter 11: Related Work

Figure 11.1:	Shared heap vs. separate heaps	139
Figure 11.2:	Stacks with mixed frames	140

Chapter 12: Evaluation

Figure 12.1:	Components structure and simplified object graph of the file system	147
Figure 12.2:	IOZONE: JX vs. Linux	149
Figure 12.3:	NFS server configuration and request rate	150
Figure 12.4:	Database benchmark results	152
Figure 12.5:	IOZONE performance	154
Figure 12.6:	IOZONE performance:Stack size check V2 vs. V1	155
Figure 12.7:	multidomain with null monitor	157
Figure 12.8:	multidomain with access check at every read/write	157
Figure 12.9:	Call graph	158
Figure 12.10:	Method timing of RPC processing	162
Figure 12.11:	Application-generated events	163
Figure 12.12:	Thread activity diagram and P6 performance counters	164
Figure 12.13:	Object allocations over time	165
Figure 12.14:	Object lifetimes	166
Figure 12.15:	Age distribution of class instances	167

Chapter 13: Conclusions

List of Tables

Chapter 1: Introduction

Chapter 2: Background

Chapter 3: Architectural Overview

Tab. 3.1	Performance of simple Java operations	24
Tab. 3.2	Event logging overhead	24

Chapter 4: Domains

Tab. 4.1	Domain creation time and destruction time	31
----------	---	----

Chapter 5: Portals

Tab. 5.1	IPC latency (round-trip, no parameters)	52
----------	---	----

Chapter 6: Components

Chapter 7: Processor Scheduling

Chapter 8: Memory Management

Tab. 8.1	Virtual method invocation with different stack size checks	96
Tab. 8.2	Performance of memory operations	96
Tab. 8.3	Memory creation time with and without the optimization <i>delayed MCB creation</i>	98
Tab. 8.4	Memory access time of mapped memory vs. get/set	98

Chapter 9: Device Drivers

Chapter 10: Security Architecture

Tab. 10.1	Operating system code sizes	134
-----------	-----------------------------	-----

Chapter 11: Related Work

Chapter 12: Evaluation

Tab. 12.1	Web server request rate	152
-----------	-------------------------	-----

Chapter 13: Conclusions

Acknowledgements

Building a complete operating system is nothing one person can accomplish alone. Many people have contributed to the JX project. JX grew out of my master's thesis about a meta architecture for Java, called MetaJava and later metaXa [73]. For this thesis I built a JVM and obtained first experiences with JVM internals. After I finished this thesis in 1997 I was employed as a research and teaching assistant at the Department of Computer Science (Operating Systems) of the University of Erlangen. Giving lectures and seminars allowed me to tell students about my ideas and convince them to do their thesis work in the context of the JX project. The first of those students was Hans Kopp who built a bytecode-to-x86 translator for metaXa as part of his master's thesis in 1998 [106]. Hans' compiler did considerably improve the performance of metaXa and using it as the foundation of an operating system became practical. At the end of 1998 I modified metaXa to run on the bare hardware and called it metaXaOS. At the same time I started to implement a new virtual machine, called JX, that provided multiple protection domains. In 1999 Andreas Weissel ported the IDE driver and the ext2 file system from Linux to metaXaOS [186]. Markus Meyerhöfer ported the driver for the 3Com 3C905B network interface card from Linux to JX and started to implement a TCP protocol stack [127]. In 2000 Christian Wawersich and Andreas Heiduk started to work on their master's thesis. Christian restructured Hans' translator and improved the quality of the generated machine code [185]. Christian's compiler allowed to run JX operating system components nearly as fast as an operating system written in C. Andreas wrote a driver for the Matrox G200 video card and the Brooktree BT878 frame grabber device [88] and a fascinating demo - an interactive puzzle of a live video source. At the end of 2000 Martin Alt and Meik Felser started their master's thesis. Martin wrote a bytecode verifier that performed the standard bytecode verification but could additionally be used to verify worst case execution times [5]. Using Martin's verifier we were able to guarantee an upper bound for response times without manually inspecting each interrupt handler. Meik focused on scheduling and multiprocessor support [63]. He ported JX to an SMP architecture and designed and implemented an interface to use Java schedulers. In 2001 Jürgen Obernolte, Christian Flügel, and Ivan Dedinski started to work on their semester thesis. Jürgen implemented a window manager [138]. Christian implemented a client for the Remote Desktop Protocol (RDP) [65] and Ivan implemented a database system [47]. Jörg Baumann restructured the garbage collector to allow domain-specific garbage collection and improved the performance visualization system. In 2002 Marco Winter implemented the AWT based on Jürgen's window manager [188]. I am grateful for the opportunity to work with all these talented students. Without them I would not have been able to prove that my design leads to a usable operating system.

I am deeply indebted to Jürgen Kleinöder who supported me from the beginning of the project. I thank Franz Hauck and the other members of the Ergoo group for the critical comments after my presentation dry-runs. This led to considerable improvements of the final presentations. I thank Fridolin Hofmann and Bernd Hindel for referring my thesis and providing valuable comments and Wolfgang Schröder-Preikschat and Heinz Gerhäuser for being members of my committee.

Finally I owe thanks to my parents, my wife Anja, and my children Tobias and Rebecca for their patience and constant support during the often exhausting process of writing this dissertation.

