The JX Operating System: An Overview

Michael Golm

University of Erlangen
Department of Computer Science
(Operating Systems and Distributed Systems)
Martensstrasse 1
91058 Erlangen, Germany

golm@cs.fau.de
Objectives of the JX system

- Make writing an OS as easy as writing applications
  - simple and robust architecture

- Dynamic OS extension with untrusted components
  - exact resource accounting and customizable management

- Code reuse
  - tailored OS configurations; dedicated systems

- Protection
  - flexibility
  - performance
  - robustness
Domains

- Domain is unit of
  - protection
  - resource management
  - fault containment
  - termination

*Single Address Space*
Components contain 100% Java

- Bytecode is verified by an extended verifier

- All components are compiled to native code at load time

DomainZero contains core written in C and assembler
Objects & Heap

- No shared objects
  - decoupled garbage collection
  - no uncontroled information flow

Single Address Space
No shared threads
⇒ decoupled CPU scheduling
⇒ two-level scheduling
Domain can export a service associated with interface component and thread.
Domain can obtain a Portal to the service

Portals are capabilities and can be passed between domains

Single Address Space
Portal call:

- threads are switched (rendezvous, handoff)
- object arguments: copy
- portal arguments: new portal to same service

Single Address Space
Advantages
- access control possible
- can create subranges
- revocation

Zero-copy components (e.g. network stack)

Single Address Space
Device Driver

Interrupt ➔ Invocation of a registered portal (1st-level interrupt handler)

Single Address Space
Protection: Interrupt Handler

- Interrupts on the interrupted CPU are blocked during execution of the interrupt handler

- Verifier checks interrupt handler for upper limit of execution time
  - can insert runtime checks to ensure timely termination
  - runtime check can terminate interrupt handler and initiate counter measure (e.g., switch off device interrupts)
Protection: Device Driver

- JX protection is based on type safety and portals
- Some domains can circumvent these mechanisms
  - DomainZero, Translator, Verifier ➔ Trust
  - (some) device drivers

---

Diagram:
- Device Driver
  - not trusted (complex driver)
  - trusted (e.g. DMA initialization, switch off device interrupts)
Building an OS: A Dedicated System

Components
- NFSd
- VFS
- RPC
- ext2FS
- UDP
- Buffer
- IP
- Ether
- NIC
- IDE

NFSDomain

DomainZero

Dedicated NFS Server

Network

Disk
Building an OS: A Multiuser System

Components

DomainZero

UserAppl A

NFS Daemon

Network

Multi-User System

Disk

Network
Building an OS: Extensibility

Components

UserAppl A

FS

???

NFS Daemon

FS

NFSd

RPC

UDP

FileSystem

FS

VFS

???FS

ext2FS

Buffer

BlockIO

Network

UDP

IP

Ether

NIC

BlockProvider

BlockIO

IDE

Disk

Multi-User System with Extensible Block Mgt

DomainZero

Buffer

BlockIO

Filesystem

Extensible Block Mgt

FS

NFSd

RPC

UDP

IDE

Disk

Network
Building an OS: Extensibility

Components

Extensible Block Mgt

DomainZero

UserAppl A

NFS Daemon

Multi-User System with Extensible Block Mgt

Network

VFS

FS

???

ext2FS

Buffer

BlockIO

BlockProvider

File System

UDP

IP

Ether

NIC

IFS

BlockIO

Distr.BlockMgr

UDP

RPC

FS

NFSd

UDP

Buffer

BlockIO
Performance

- Hardware:
  500MHz PIII, 128MB RAM,
  IDE: Maxtor 91303D6, 12427MB, 512kB Cache
  NIC: 3C905B 100 MB/s

- IPC:
  - Portal round trip 650 cycles
  - L4 (including RPC stubs): 800 cycles
  - KaffeOS: 27270 cycles
Performance

- lozone-like benchmark: re-read

<table>
<thead>
<tr>
<th>JX</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>file size (kB)</td>
<td>throughput (kB/s)</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>
Performance

- Iozone-like benchmark: re-read of a 512 kB file
Performance

JX as NFS server: `getattr` request rate

![Graph showing `getattr` request rate for JX and Linux over time. The graph displays two lines, one for JX and one for Linux, indicating different request rates at various times.]
Performance: JX Advantages and Limitations

■ Advantages:
  • No expensive border crossings (JNI, OS border)
  • Safe inlining of OS-level code into application code
  • Avoid locking in favour of specialized scheduling

■ Limitations:
  • Unavoidable safety checks
  • Semantic gap between stack machine and register machine
Conclusion

■ Single Address Space

■ Full Protection
  • completely decoupled domains
  • fast communication using portals or memory objects

■ Reusable components

■ Dynamic extensibility

■ Good performance

→ http://www4.cs.fau.de/Projects/JX/
Status

- JX runs on off-the-shelf PCs
- Drivers for: IDE, Matrox G200, 3COM 3C905B, BT848
- Ext2-FS
- UDP, TCP, RPC (client), NFS (client)
- SMP support
Component Interaction

<table>
<thead>
<tr>
<th></th>
<th>co-located components</th>
<th>dis-located components</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter passing</td>
<td>by reference possible</td>
<td>parameter objects must be copied</td>
</tr>
<tr>
<td>thread</td>
<td>can execute in same thread</td>
<td>threads must be switched</td>
</tr>
<tr>
<td>resources</td>
<td>caller can be trusted to carefully use resources (e.g., Memory objects)</td>
<td>access rights for Memory objects must be restricted</td>
</tr>
</tbody>
</table>

- Scheduling: scheduling strategy of all co-located components must be compatible

- Execution engine of co-located components must be identical (translated or interpreted)
Memory Mapping

- Performance problem: range check
- (partial) solution: map objects to memory range
Memory Mapping

- Performance problem: range check
- (partial) solution: map objects to memory range

```
int a
byte b
int c
char d
class X
```
Memory Mapping

- Performance problem: range check
- (partial) solution: map objects to memory range
Scheduling

Domain A
- Components
- Heap
- Threads
- Java Stacks
- Control Blocks

Domain B
- Components
- Heap
- Threads
- Java Stacks
- Control Blocks

Domain C
- Components
- Heap
- Threads
- Java Stacks
- Control Blocks

Domain Zero
- C Assembler
- Native Stacks
- Threads
- Control Blocks

LowLevelScheduler

Domains

CPU

Domains

CPU
Scheduling

- Domain A
  - Components
  - Heap
  - Threads
  - Java Stacks
  - Thread Control Blocks

- Domain B
  - Components
  - Heap
  - Threads
  - Java Stacks
  - Thread Control Blocks

- Domain C
  - Components
  - Heap
  - Threads
  - Java Stacks
  - Thread Control Blocks

- Domain Zero
  - Assembler
  - Threads
  - Native Stacks
  - Thread Control Blocks

- HighLevelSchedulerA
- HighLevelSchedulerB
- HighLevelSchedulerC

- LowLevelScheduler

- CPUs
Resource Management

- Physical resources (CPU, Memory)
  - resource management is supported by DomainZero (but no policy)

- Device-specific physical resources (e.g. network bandwidth)
  - completely managed by a domain

- Virtual resources
  - managed by their respective domains (e.g. TCP port numbers, files)
Hardware Access

Domain A

Components

Heap

Java Stacks

Threads

Thread Control Blocks

direct access to hardware registers

Domain B

Components

Heap

Java Stacks

Threads

Thread Control Blocks

Single Address Space

Disk

Network

RAM

Clock

CPU

Video

Keyboard

C Assembler

Native Stacks

Thread Control Blocks

Domain Zero

JX Architecture
Zero-copy using Memory

- immediate processing or buffering possible

upstream:

```
mem1 = receive(mem0)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)
```

```
mem0 = processMemory(mem1)