The JX Operating System

Michael Golm
Meik Felser, Christian Wawersich, Jürgen Kleinöder

University of Erlangen-Nürnberg
Department of Computer Science
(Distributed Systems and Operating Systems)
Martensstraße 1
91058 Erlangen, Germany
golm@cs.fau.de
A Dancing Bear

This is a "dancing bear" paper -- it's not how well the bear dances, but that it dances at all.
And this bear dances.

Anonymous Reviewer
Outline

■ Motivation

■ JX Architecture
  ◆ Protection domains
  ◆ Communication mechanism
  ◆ The Microkernel

■ System-level programming in Java

■ Performance
Abstraction levels in software engineering

- Assembler
  - portability
  - productivity
  - robustness

- C
  - portability
  - productivity
  - robustness

- Smalltalk, Java, C#
Why another OS?

Abstraction levels in software engineering

Assembler

C

portability
productivity
robustness

Smalltalk, Java, C#

Applications

Operating Systems

Motivation
Typical System Structure

Application

Middleware (JVM, RMI, CORBA,...)

OS Kernel (Unix,...)

Hardware
Typical System Structure

Application

JVM

Unix

Hardware
Typical System Structure

**Application**
- File Object

**Java Native Interface**

**JVM**
- File Descriptor

**System Call Interface**

**Unix**
- Disk Driver
- File System

**Hardware Interface**

**Hardware**

**The JX Operating System**
The Way to a new Architecture

Motivation

The JX Operating System

Application
  File Object

Java Native Interface

JVM
  File Descriptor

System Call Interface

Unix
  Disk Driver

Hardware Interface

Hardware
  File System
The JX Operating System

Motivation

The Way to a new Architecture

Application
File Object

File System
File System

Device Driver
Disk Driver

JVM

System Call Interface

OS Kernel (μKernel)

Hardware Interface

Hardware

File Descriptor

Disk Driver

File System

The JX Operating System
The JX Operating System

Motivation

The Way to a new Architecture

Application
File Object

File System
File System

Device Driver
Disk Driver

JVM (w/o native methods)
File Descriptor

System Call Interface

OS Kernel (µKernel)
Disk Driver
File System

Hardware Interface

Hardware

The JX Operating System
The Way to a new Architecture

JVM (w/o native methods)

OS Kernel (\(\mu\)Kernel)

Hardware Interface

Hardware
The JX Operating System

- Application
  - File Object
- File System
  - File System
- Device Driver
  - Disk Driver

JX (μKernel)

Hardware Interface

Hardware
The JX Operating System

Architecture

Application

File Object

File System

File System

Device Driver

Disk Driver

JVM

JVM

JVM

JX (μKernel)

Hardware Interface

Hardware
Protection Domains

{diagram showing protection domains with virtual JVMs and a JX (μKernel) at the center, connected to hardware through a hardware interface}
Protection Domains

Domain

Components

Threads

Heap

JX (µKernel)

Hardware Interface

Hardware

The JX Operating System
The JX Operating System

Communication: Portals

Architecture

Domain

Components

Heap

Threads

JX (µKernel)

Hardware Interface

Hardware

Domain

Components

Heap

Threads

Service

 Domain

 Domain

 The JX Operating System
Communication: Portals

Architecture

Domain

Components

Heap

Threads

JX (µKernel)

Hardware Interface

Hardware

Domain

Components

Heap

Threads

Service

Implementation

Interface
Communication: Portals

Domain

Components

Heap

Threads

JX (µKernel)

Hardware Interface

Hardware
Communication: Portals

JX (\(\mu\)Kernel)

Hardware Interface

Hardware
Communication: Portals

Architecture

Domain

Portal

Heap

Threads

Components

Interface

JX (μKernel)

Hardware Interface

Hardware

Domain

Service

Heap

Service-Thread

Threads

Components

Implementation

Interface
Communication: Portals

Domain

Portal

Heap

Client-Thread Threads

Components

Interface

Service

Heap

Service-Thread Threads

Implementation

Interface

Components

JX (μKernel)

Hardware Interface

Hardware

The JX Operating System 22
Protection Domains

- Each domain has its own heap
  - no shared objects → no accounting problems (memory, GC time)
  - no GC dependencies → no scalability problems
  - explicit application boundaries → confinement

- Each domain has its own threads
  - no migrating threads → no domain termination problems

- Each domain has its own code
  - no trusted code → improved security
The Microkernel

Architecture

The JX Operating System
The Microkernel

DomainZero

- Java-level collector
- copying garbage collector
- mark&sweep garbage collector
- garbage collector support framework
- domain management
- portal invocation
- memory management
- component management
- low-level CPU management
- preemptive round-robin scheduler
- non-preemptive round-robin scheduler
- Java-level scheduler
- low-level CPU management
- monitoring

Runtime System

Portals

Memory Manager

Domain Manager

Naming
Outline

■ Motivation

■ JX Architecture
  ◆ Protection domains
  ◆ Communication mechanism
  ◆ The Microkernel

■ System-level programming in Java

■ Performance
System-level Programming in Java

Application
- File Object

File System
- File System

Device Driver
- Disk Driver

JX

Hardware Interface

Hardware

The JX Operating System
System-level Programming in Java

■ Problems
  ♦ Management of large memory, registers, and on-device memory
  ♦ Interrupt handlers
System-level Programming in Java

■ Problems
  ◆ Management of large memory, registers, and on-device memory
  ◆ Interrupt handlers

■ Requirement
  ◆ No changes to the language or bytecode instruction set
Memory

- Manage large memory areas
  - disk blocks, network buffers, ...

- Access special memory areas
  - video memory, device registers, ...

- Disadvantages of arrays:
  - no explicit interface
  - no specializations possible
  - located on heap, can be moved (DMA!)
  - can only be passed by copying
Memory Portals

- Explicit interface
- Can be subtyped
- Can be used to access arbitrary memory areas
- Can be treated specially by the translator
- Share data between domains
- Pass subranges to other domains
- Revoke access
Fast Portals

- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level
Fast Portals

- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level
- Example: Memory portal

```
Client Domain

heap

memoryManager.alloc(...)

DomainZero
```
Fast Portals

- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level
- Example: Memory portal

```
Client Domain

memoryManager.alloc(...)

heap

DomainZero

allocate memory

memory area
```
Fast Portals

- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level
- Example: Memory portal

```
Client Domain

memoryManager.alloc(...)

heap

mem
size...

 DomainZero

allocate memory

memory area

copy portal to client
```
Interrupt Handler

- Interrupts handled in own thread with interrupts disabled
  - must not run in endless loop ➔ verifier
  - must not invoke portals except fast portals
  - must synchronize with other threads without blocking

▲ AtomicVariable

- `setValue(...)`, `unblock(...)` used in interrupt thread
- `blockIfEqual(...)` used outside interrupt thread

➔ Single Producer / Single Consumer Queue
System-level Programming in Java

Device Driver Domain

- Threads
- IRQ-Thread
- Heap
- AtomicVariable

Memory Regions:

- DRAM
- Heap
- Registers, On-device memory
System-level Programming in Java

Device Driver Domain

- Threads
- IRQ-Thread
- Heap
- AtomicVariable
- DeviceMemory
- Memory

Layers:
- DRAM
- Memory
- Heap
- Registers, On-device memory
Outline

■ Motivation

■ JX Architecture
  ◆ Protection domains
  ◆ Communication mechanism
  ◆ The Microkernel

■ System-level programming in Java

■ Performance
  ◆ IOZONE
  ◆ NFS
IOZone: Multi-domain Config

The JX Operating System
IOZone: Multi-domain Config

Performance

The JX Operating System
IOZone: Single-domain Config

Performance

Filesystem

BlockProvider

IOZone

JX vs. Linux

achieved throughput in percent

filesize in KBytes

record size in KBytes

The JX Operating System
IOZone: Single-domain Config

Performance

IOZone

FS
ext2FS
Buffer
BlockIO
IDE
DomainZero
BlockProvider

Filesystem

FS
Buffer
BlockIO
IDE

Running without safety checks

Improvement in percent

Record size in KBytes

Filesize in KBytes

4 8 16 32 64 128 256 512

The JX Operating System
NFS: *getattr* request rate

- Linux (kernel NFS)
- Linux (user NFS)
- JX

Graph showing request rate over time for different systems.
NFS: *getattr* request rate
**Performance**

**NFS: `getattr` request rate**

- Idle 0.1
- DomainZero:InitialThread 0.3
- SVC-jx/zero/BootFS 0.4
- SVC-jx/zero/ComponentManager 0.5
- SVC-jx/zero/DebugSupport 0.6
- SVC-jx/zero/DomainManager 0.7
- SVC-jx/zero/DebugSupport 0.8
- SVC-jx/zero/Naming 0.9
- Init-main 1.2
- GC 2.1
- NFS-main 2.2
- jx/devices/pci/PCIGod 2.3
- timerpc/StartTimer 2.4
- jx/net/StartNetDevice 2.5
- jx/net/protocols/StartNetworkProtocols 2.6
- test/fs/IDEDomain 2.7
- FSDomain-Main 2.8
- test/nfs/NFSDomain 2.9
- SVC-jx/devices/pci/PCIAccess 2.10
- TimerManager 2.11
- IRQThread 2.12
- SVC-jx/timer/TimerManager 2.13
- IRQThread 2.14
- IDE-2nd-IRQ ide0 2.15
- IDE-2nd-IRQ ide1 2.16
- IRQThread 2.17
- SVC-jx/devices/net/NetworkDevice 2.18
- Etherpacket-Queue 2.19
- SVC-jx/net/NetInit 2.20
- RPC-Receiver 2.21
- SVC-bioide/Partition 2.22
- SVC-jx/fs/FS 2.23
- SVC-jx/fs/FileSystem 2.24
- MountProc 2.26
- NFSProc 2.27
- RPCBind 2.28
NFS: *getattr* request rate
NFS: *getattr* request rate

The JX Operating System

Performance

Time in Seconds

0 2 4 6 8 10 12 14 16 18 20 22
Conclusion

- It is possible to build an OS based on type-safe protection
  - Isolation of protection domains
  - System-level programming without language extensions

- OS takes advantage of advanced language technology
  - Increased robustness of system components
  - Modular system
  - Flexibility in system configuration
  - Extensibility using domains

- Performance is sufficient for most applications
  ➔ [http://www4.cs.fau.de/Projects/JX](http://www4.cs.fau.de/Projects/JX)