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
  - tailored OS configurations; dedicated systems

- Protection
  - flexibility
  - performance
  - robustness
  - code reuse

- Code reuse

- Dynamic OS extension with untrusted components
  - simple and robust architecture

- Make writing an OS as easy as writing applications
Domain Zero contains core written in C and assembler. Domain Zero contains core native code at load time.

- All components are compiled to
- Bytecode is verified by an extended verifier
- Components contain 100% Java

- Components are compiled to native code at load time

Domain is unit of
- Protection
- Resource Management
- Fault Containment
- Termination
- Domain is unit of

Single Address Space
Objects & Heap

Domain Zero

C

Assembler

Domain A

Components

Classes

Heap

Domain B

Components

Classes

Heap

Threads

Domain Zero

Assembler

Domain A

Components

Classes

Threads

Domain B

Components

Classes

Threads

Single Address Space

No shared objects ➔ decoupled garbage collection ➔ no uncontrolled information flow

No shared threads ➔ decoupled CPU scheduling ➔ two-level scheduling

JX Architecture

Objects & Heap

JX Architecture

Threads

Objects & Heap

JX Architecture
Domain can export a service
- Portals are capabilities and can be passed between domains
- Domain can obtain a Portal to the service

Domain can export a service
- Portals are capabilities and can be passed between domains
- Domain can obtain a Portal to the service

Portals: Components and threads associated with an interface

JX Architecture: Portals
JX Architecture

Communication: Portals

Domain Zero
C
Assembler

Domain A
Components
Classes
Heap
Portal
Threads

Domain B
Components
Classes
Heap
Portal
Threads

Portals: Portals

Advantages
- Zero-copy components
- Recognition
- Can create subranges
- Access control possible

Single Address Space

Zero-copy components

Memory

JX Architecture: Memory

Advantages
- Access control possible
- Can create subranges
- Revocation

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

Zero-copy components:
- New portal to same service
- Copy (rendezvous, handoff)
- Threads are switched

Portal call:

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. Runtime check can insert timed termination of interrupt handler. Runtime check can terminate interrupt handler and initiate countermeasure (e.g., switch off device interrupts).

Verifier checks interrupt handler for upper limit of execution time. Runtime check can insert timed termination of interrupt handler. Runtime check can terminate interrupt handler and initiate countermeasure (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
  - (some) device drivers

Building an OS: A Dedicated System

Component

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

Device Driver

- Trust (e.g., DMA initialization, switch off device interrupts)
- Not trusted (complex drivers)

- Trusted device drivers
- DomainZero, Translator, Verifier
  - Trust

- Some domains can circumvent these mechanisms
  - JX protection is based on type safety and portals
### Hardware
- 500MHz PIII, 128MB RAM
- IDE: Maxtor 91303D6, 12427MB, 512kB Cache
- NIC: 3C905B 100 MB/s

### IPC
- KaffeOS: 27270 cycles
- L4 (including RPC stubs): 800 cycles
- Portal Round Trip: 650 cycles

### Performance
- Iozone-like benchmark: re-read

<table>
<thead>
<tr>
<th>File Size (kB)</th>
<th>Throughput (kB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>512</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>512</td>
<td>128</td>
</tr>
<tr>
<td>1024</td>
<td>256</td>
</tr>
<tr>
<td>2048</td>
<td>512</td>
</tr>
<tr>
<td>4096</td>
<td>1024</td>
</tr>
<tr>
<td>8192</td>
<td>2048</td>
</tr>
</tbody>
</table>

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

Iozone-like benchmark: read of a 512 KB file

JX as NFS server: getattr request rate

Iozone-like benchmark: re-read of a 512 KB file
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
• Reusable components
• Full Protection
• Fast communication using portals or memory objects
• Completely decoupled domains
• Good performance
• Dynamic extensibility
• Reusable components

http://www4.cs.fau.de/Projects/JX/
**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
- Drivers for: IDE, Matrox G200, 3COM 3C905B, BT848
- Runs on off-the-shelf PCs

### Component Interaction

<table>
<thead>
<tr>
<th>Flexibility</th>
<th>Scheduling</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-located Components</td>
<td>Parameter passing</td>
<td>Memory objects</td>
</tr>
<tr>
<td>Dis-located Components</td>
<td>Parameter passing</td>
<td>Fully use resources (e.g., access rights for memory objects)</td>
</tr>
<tr>
<td></td>
<td>Switched threads must be</td>
<td>Can execute in same thread</td>
</tr>
<tr>
<td></td>
<td>Scheduling strategy of all co-located components must be identical</td>
<td>Caller can be trusted to care</td>
</tr>
</tbody>
</table>

**Status**
Memory Mapping

Performance problem: range check

(partial) solution: map objects to memory range

Memory Mapping

Memory

Performance problem: range check

(partial) solution: map objects to memory range
Memory Mapping

* Partial solution: map objects to memory range

* Performance problem: range check
Resource Management

Physical resources (CPU, Memory)
- Device-specific physical resources (e.g., network bandwidth)
  - Completely managed by a domain (but no policy)
- Virtual resources
  - Managed by their respective domains

Resource management is supported by DomainZero

Scheduleing
Zero-copy using Memory

- immediate processing or buffering possible

upstream:

mem0 = processMemory(mem1)

mem0 = processMemory(mem1)

mem0 = processMemory(mem1)

mem0 = processMemory(mem1)

mem0 = processMemory(mem1)

DMA

NIC