Falling in Love with EROS
(Or Not)

Robert Grimm
New York University
The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?
Basic Access Control

- Access control lists
  - Are associated with objects
  - Specify which subjects can perform which operations

- Capabilities
  - Are associated with subjects
  - Specify which operations can be performed on which objects
Build a Secure Service

- Service performs operations for different subjects
  - Using access control lists
    - Create lots of user IDs, change service’s ID on demand
      - Hard to configure/manage; does not work well with threading
    - Make service all powerful, enforce access control in service
      - Common strategy today; but also violates least privilege
  - Using capabilities
    - Just pass necessary capabilities to service...
Challenges with Capabilities

- Bootstrapping
  - How to hand out (initial) capabilities?
- Traceability, revocation, extensibility
  - How to audit and change rights?
  - How to enforce higher-level policies?
- Performance
  - How to make it fast?
Eros in a Nutshell

* Bootstrapping
  * Single level store
* Traceability, revocation, extensibility
  * Version number for objects and capabilities
  * Weak capabilities: degrade to read-only, no-call
  * A single system call: capability invocation
* Performance
  * Stay close to hardware: pages and nodes
  * Optimize the heck out of address translation
  * Rely on Jochen Liedtke for IPC optimization
Kernel Design
Kernel Abstractions

- Numbers to capture register values
- Nodes to hierarchically organize state
  - Machine independent state
- Data and capability pages for large data items
- Processes to execute code
- Entry and resume capabilities to transfer control
Address space: vpage $\rightarrow$ ppage $\times \{r,w\} \times$ handler (?)

**Figure 2.** EROS memory tree
Processes really are kernel-level threads

- Directly store state in nodes, as capabilities

**Figure 3.** Process structure
A single system call

- 4 data registers
- 1 contiguous data string
- 4 capability registers
  - Last capability may be a “resume capability”
Persistence

Basic approach: periodically snapshot all system state

Actual implementation

- Perform consistency check
  - Pointers point to objects of right types
  - Read-only objects have not changed
  - Modified objects listed in checkpoint directory
  - Types of capabilities in process slots are valid
- Mark system as copy-on-write
- Asynchronously write out system state
Kernel Implementation
Capabilities

- On-disk representation
  - Prepared representation
    - Object in memory
    - Capability is valid
    - Capability points to object
    - (Object linked with capability)
Address Translation

- Reduce traversal cost
  - Do not allocate intermediate nodes if only slot 0 is used
  - Track nodes (hard state) producing page table (soft state)
- Share mapping tables
  - Share nodes between process and resulting tables as well
- Avoid inverted page tables
  - Leverage existing doubly-linked lists
Persistence

- Two kernel design rules
  - All state resides in pages and nodes
  - All kernel operations are atomic
- But kernel invocations may stall (e.g., page I/O)
  - Adjust PC to retry invocation
  - Place (wake-up) capability on in-kernel stall queue
Jochen Liedtke to the Rescue (SOSP ’93)
“There is no single trick to obtaining this high performance; rather, a synergistic approach in design and implementation on all levels is needed.”
System calls: “call & wait” + “reply & receive”
  * Scheduling is the same for requests and replies
* Rich message structure: direct & indirect objects
  * Again, symmetry for send and receive operations
* Single copy through temporary mapping
  * Map target’s region into source’s address space
* Kernel stack per thread
  * Interrupts, page faults, IPC, etc. just use current stack
* Thread control blocks (TCBs) held in virtual memory
  * In one large array, mapped into all address spaces
Algorithmic Level

- Thread user ID structure
  - Trivially map back to control block
- Unlink TCBs from queues when unmapping
  - Otherwise, we get page faults...
- Optimized timeout bookkeeping
  - \( n \) unordered wakeup lists, \((\text{base} + \text{offset})\) for time
- Lazy scheduling
  - Avoid updating queues on IPC
- Direct process switch
  - Donate time slice to callee (as in LRPC)
- Pass short messages in register
Coding Level

- Reduce cache misses
  - E.g., use registers, short jumps and address displacements
- Reduce TLB misses
  - E.g., place all IPC code on one page, all tables on one page
- Optimize use of segment registers
- Make best use of general registers
- Avoid jumps and checks
  - E.g., jumps not taken are cheaper
- Minimize switch activities
  - E.g., only save floating point registers when actually used
Take That Mach...

Figure 8: 486-DX50, L3 versus Mach Ipc Times
Back to Eros
Basic User-Space Services

- Storage allocation through hierarchy of space banks
  - Allocate, track, and invalidate nodes and pages
  - Provide storage locality through contiguous extents on disk
- Copy-on-write through “Virtual Copy Space” handler
  - Relies on fast page fault handler for neighboring pages
- Process initialization through “constructor”
  - Inspect initial capabilities to certify code
## Measured Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Linux-Normalized</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipe Latency</td>
<td>5.66 µs</td>
<td>32.3%</td>
</tr>
<tr>
<td></td>
<td>8.34 µs</td>
<td></td>
</tr>
<tr>
<td>Pipe Bandwidth</td>
<td>281 MB/s</td>
<td></td>
</tr>
<tr>
<td></td>
<td>260 MB/s</td>
<td>8.07%</td>
</tr>
<tr>
<td>Create Process</td>
<td>0.664 ms</td>
<td>65.3%</td>
</tr>
<tr>
<td></td>
<td>1.92 ms</td>
<td></td>
</tr>
<tr>
<td>Ctxt Switch</td>
<td>1.19 µs</td>
<td>5.5%</td>
</tr>
<tr>
<td></td>
<td>1.26 µs</td>
<td></td>
</tr>
<tr>
<td>Grow Heap</td>
<td>20.42 µs</td>
<td>35.7%</td>
</tr>
<tr>
<td></td>
<td>31.74 µs</td>
<td></td>
</tr>
<tr>
<td>Page Fault</td>
<td>8.67 µs</td>
<td>99.5%</td>
</tr>
<tr>
<td></td>
<td>687 µs</td>
<td></td>
</tr>
<tr>
<td>Trivial Syscall</td>
<td>1.6 µs</td>
<td>-128%</td>
</tr>
<tr>
<td></td>
<td>0.7 µs</td>
<td></td>
</tr>
</tbody>
</table>
What Do You Think?