Microkernel Construction
Capabilities

Nils Asmussen

06/22/2017
Outline

- Introduction
  - Global Names
  - ACL
- Capabilities in General
- Capabilities in NOVA
Motivation

- How do you find/access resources?
- How do you restrict access to resources?
Global Names

- One global namespace for (one type of) resources
- Example: semaphores, processes, devices, ... on UNIX

Pros & Cons

+ Simple
- Name clashes: people need to agree on names.
- What if a malicious process registers a name first?
- All resources are visible: just try to access them
Access Control Lists

- Attach a list of permissions (subjects) to each object
- Permission depends on who you are, not what you have

<table>
<thead>
<tr>
<th>Pros &amp; Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ No need to give permissions explicitly</td>
</tr>
<tr>
<td>+ Makes it easy to restrict access to specific objects</td>
</tr>
<tr>
<td>- Makes it hard to restrict specific subjects</td>
</tr>
<tr>
<td>- POLA is more difficult to achieve</td>
</tr>
<tr>
<td>- Requires global names</td>
</tr>
<tr>
<td>- Confused deputy problem</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Capabilities in General
  - Overview
  - Operations
- Capabilities in NOVA
Capabilities

- Give each subject a local namespace
- Operations to exchange objects between namespaces
- Permission depends on what you have

Pros & Cons

+ Makes it easy to restrict specific subjects
+ Separation of subsystems, composable, independent
+ POLA is easy to achieve
  - Need to give permissions explicitly
  - Exchanging, especially revoking, capabilities is difficult
Overview

Selector | Capability
---|---

Process A

Process B

Selector | Capability
---|---

K. Object | Cap Space
---|---

Thread | Sem | Process | File | Thread
---|---|---|---|---
Map/delegate:
- Copy capability from one Cap Space to the other
Grant:
- Move capability from one Cap Space to the other
Revoke:
- Remove capability, recursively
Lookup:
- Search capability by selector and return its permissions
Translate:
- Translate selector from one Cap Space to the other
Hierarchical Organization

- Microkernel
- Root Task
- Pager 1
- Pager 2
- Pager 3
- Application
- Application
- Phys. memory 1-to-1 mapped
- Root Task
- Microkernel
Outline

- Introduction
- Capabilities in General
- Capabilities in NOVA
  - Capability Spaces
  - Mapping Database
  - Delegate, Translate and Revoke
  - Data Types
  - Receive Windows
Each protection domain (Pd) has

- Space_obj: object capabilities
- Space_mem: memory capabilities (pages)
- Space_pio: I/O port capabilities

Similarities and differences

- Shared: capability delegation, revocation, ...
- Differences:
  - Object caps are created and used via system calls
  - Port and memory caps are referring to existing resources
  - Passed to root task, distributed in the system via delegation
  - Memory capabilities lead to page table entries
  - Port capabilities lead to bits set in the I/O bitmap
Object Capability Space

Cap Table  Mapping DB  Kernel Obj.

syscall

on create/update

on create
Memory Capability Space

Page Table → Mapping DB → Phys. Memory

ld/st

on create/update
I/O Capability Space

IO Bitmap

Mapping DB

IO ports

in/out

on create/update
Mapping Database
Mapping Database – Revoke
Mapping Database – Revoke
Order specifies the number of capabilities ($2^{order}$)

Selector specifies the first capability

Selector has to be size aligned, i.e., a multiple of $2^{order}$

**Wrong**: order=2, selector=6, okay: order=2, selector=8

Mask allows to reduce permissions

T specifies capability space (objects, memory, I/O)
Receiver sets up receive window (writes CRD into UTCB)
Receivers waits for IPC
Sender puts typed item into UTCB
Sender calls portal
Kernel delegates typed item
Kernel puts typed item into UTCB, telling receiver about caps
Kernel switches to receiver
But: what if receive window and sent caps don’t match?
Figure: Send window is smaller than receive window
Figure: Send window is larger than receive window
Deleting Mapping DB Nodes and Kernel Objects

- When revoking, kernel objects should be destructed
- But what if somebody accesses them at the same time?
- We could lock them during each access
- But this is expensive
- We don’t care that much when exactly they are destructed
- Can’t we destruct them if nobody accesses them anymore?
- Basically: copy-on-write with lazy delete
- Don’t change objects, but copy them and change the copy
- Don’t delete objects immediately, but when readers are done
- In case of NOVA: no copy-on-write, but only lazy delete
- On revoke, object is removed first
- Then, the object is registered for deletion
- Timer IRQ is used to delete only if all readers are gone
RCU Grace Period

- Remove
- Sync
- Delete

CPU 0:read
CPU 1:read
CPU 2:read
CPU 3:read
Next Weeks

- 29th June: M3
- 7th July: L4Re
- 13th July: Escape