RESOURCE MANAGEMENT

MICHAEL ROITZSCH
done: time, drivers

today: misc. resources
  - architectures for resource management
  - solutions for specific resources
  - capabilities to manage resource access

upcoming: applications, legacy support
KERNEL RESOURCES
**PROBLEM**

- Kernel needs memory for its abstractions
  - Tasks: page tables
  - Threads: kernel-TCB
  - Capability tables
  - IPC wait queues
  - Mapping database
- Kernel memory is limited
- Opens the possibility of DoS attacks
- memory management policy should not be in the kernel
- account all memory to the application it is needed for (directly or indirectly)
- kernel provides memory control mechanism
- exception for bootstrapping: initial kernel memory is managed by kernel
- untyped memory in seL4
- all physical memory unused after bootstrap is represented by untyped memory capabilities
- can be granted, split or retyped
- restricted to powers of 2 (see flexpages)
- initial resource manager gets all (see $\sigma_0$)
- user code decides how to use them
- application retype UM to kernel objects
  - TCB, endpoint, CNode, VNode, frame, interrupt
  - all kernel bookkeeping for the object uses the underlying physical memory
  - no implicit memory allocation by the kernel
- retyping and splitting is remembered in capability derivation tree
  - revoking recursively destroys all derived capabilities and kernel objects
separate enforcement and management
ARCHITECTURES
low-level resource abstractions
explicit management

high-level resource abstractions
implicit management

exokernel

multiserver

resource containers

monolith
- enforcement and management implicitly tied to process abstraction

<table>
<thead>
<tr>
<th>isolation</th>
<th>accounting</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td></td>
</tr>
<tr>
<td>protection domain</td>
<td>resource container</td>
</tr>
</tbody>
</table>

- resource containers were proposed to make resource management explicit

- bags of resources assigned to subsystems
provide primitives at the lowest possible level necessary for protection

use physical names wherever possible

resource management primitives:

- explicit allocation
- exposed revocation
- protected sharing
- ownership tracking
CONSEQUENCES

- applications can use their own library OS
- library OS’es cannot trust each other
- no global management for resources
- think of a file system
  - kernel manages disk block ownership
  - each library OS comes with its own filesystem implementation
- one partition per application?
- Invariants in shared resources must be maintained.
- 4 mechanisms provided by the exokernel:
  - Software regions for sub-page memory protection, allows to share state.
  - Capabilities for access control.
  - Critical sections.
  - Wakeup predicates: code downloaded into the kernel for arbitrary checks.
Low-Level Resource Manager

Higher-Level Resource Manager

Application

Client-Libs

L4 Microkernel

works on monolithic kernels too
different abstraction levels for resources

<table>
<thead>
<tr>
<th>basic resources</th>
<th>memory, CPU, IO-ports, interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>hardware</td>
<td>block device, framebuffer, network card</td>
</tr>
<tr>
<td>compound resources</td>
<td>file, GUI window, TCP session</td>
</tr>
</tbody>
</table>
Applications can access resources on the abstraction level they need.

Servers implementing a resource can use other, lower-level resources.

Isolation allows managers to provide real-time guarantees for their specific resource.

DROPS: Dresden Real-time OPerating System
EXAMPLES
ANKH

- driver for physical network card
- built with DDE using Linux 2.6 drivers
- provides multiple virtual network cards
- implements a simple virtual bridge
LWIP

- light-weight IP Stack
- TCP/IP, UDP, ICMP
- clients can use standard BSD socket interface
- IDE driver to access hard disks
- includes disk request scheduling
- based on DDE
- provides block device
- ongoing work on USB block devices
- no real one implemented yet
- we have a tmpfs using RAM as backing store
- VPFS: securely reuse a Linux filesystem
L4Re VFS

- hierarchical name space
- connects subtrees to different backend servers
- aka mounting
- multiplexes the frame buffer
- no virtual desktops, but window merging
- details in the legacy / security lectures
Terminal

- widget drawing server
- handles mouse and keyboard input
- can also operate on raw framebuffer
- real-time capable

DOpE

mag
- DOpE client providing a terminal window
- VT100 emulation
- can support readline applications
  - shell
  - python
RESOURCE ACCESS
Worker A

Manager

Worker B

Service
- separate processes
- chrome parent
- sandboxes for tabs
- implementation on Linux: glorious mix of chroot(), clone() and setuid()
- there must be a better way…
<table>
<thead>
<tr>
<th></th>
<th><strong>POSIX</strong></th>
<th></th>
<th><strong>POLA</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>operations</td>
<td>operations allowed by default</td>
<td></td>
<td>nothing allowed by default</td>
</tr>
<tr>
<td></td>
<td>some limited restrictions apply</td>
<td></td>
<td>every right must be granted</td>
</tr>
<tr>
<td></td>
<td>ambient authority</td>
<td></td>
<td>explicit authority</td>
</tr>
</tbody>
</table>
L4Re – the L4 Runtime Environment
set of libraries and system services on
top of the Fiasco.OC microkernel
Fiasco.OC and L4Re form an object-capability system

- actors in the system are objects
  - objects have local state and behavior
- capabilities are references to objects
  - any object interaction requires a capability
  - unseparable and unforgeable combination of reference and access right
invocation of any object requires a capability to that object

- no global names

- no sophisticated rights representation beyond capability ownership

- just four rights bits on objects

- C++ language integration

- capabilities passed as message payload
Worker A

Manager

Worker B

mag
- factory for new framebuffer sessions
- session object
  - backing store memory
  - view: visible rectangle on the backing store
  - metadata, refresh method
- How does it appear on the screen?
- hardware framebuffer is memory with side effect
- all memory is initially mapped to the root task
- framebuffer driver
  - find framebuffer memory
  - wrap in FB-interface
- same interface as mag's
- virtualizable interfaces
- L4Re uses one interface per resource
  - independent of the implementation
  - servers can (re-)implement any interface
- the kernel is a special server: provides low-level objects that need CPU privileges
  - minimal policy
- userland servers can augment
EXAMPLES

Graphics

pong

mag

fb-drv

Thread scheduling

multithreaded application

balancer

kernel

Graphics

Thread scheduling

pong

mag

fb-drv

multithreaded application

balancer

kernel

MOS: Resource Management
CONCLUSION

- all services provided as objects
- uniform access control with capabilities
- invocation is the only system call
- virtualizable: all interfaces can be interposed
- resource refinement and multiplexing transparent to clients
SUMMARY

- kernel resource management
- basic resource management concepts
  - resource containers
  - exokernel
  - multiserver
- management details for specific resources
- object capabilities and virtualizable interfaces