THE FIASCO.OC MICROKERNEL
Alexander Warg
FIASCO.OC IN ONE SLIDE

CAPABILITY-BASED MICROKERNEL API
- single system call → invoke capability

MULTI-PROCESSOR SUPPORT
- tested up to 48 CPUs

X-CPU HELPING LOCKS
- avoid priority inversion (for RT software)

VERSATILE CPU AND PLATFORM SUPPORT
- IA32 (x86 and x86_64)
- ARM (v5 ... v7) (14+ platforms)
CONTENTS

FIASCO.OC

- basic concepts
- kernel API concepts
- kernel object details
- IPC details
- vCPU details
FIASCO.OC / BASIC CONCEPTS

API BASICS
TASKS (PROTECTION DOMAINS)
THREADS
IRQS
CAPABILITIES
...

MICROKERNEL CONSTRUCTION
API CONCEPTS

EVERYTHING IS AN OBJECT
- all system calls run on an object
- 'one system call' — send message to object

CAPABILITIES — THE OBJECT REFERENCES

UNIFORM INTERFACES
- kernel and user-level objects with uniform interfaces

FACTORY FOR OBJECT CREATION
CAPABILITIES

REFERENCES TO KERNEL OBJECTS
- local naming (per protection domain)
- access control

KEPT IN PER TASK CAPABILITY TABLE
- comparable to file descriptors, file-descriptor table
## OVERVIEW: KERNEL OBJECTS

<table>
<thead>
<tr>
<th><strong>TASK</strong></th>
<th>protection domain</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>THREAD</strong></td>
<td>thread of execution in a task</td>
</tr>
<tr>
<td><strong>FACTORY</strong></td>
<td>creation of (kernel) objects</td>
</tr>
<tr>
<td><strong>IPC GATE</strong></td>
<td>communication channel / object</td>
</tr>
<tr>
<td><strong>IRQ</strong></td>
<td>async. signalling (software / hardware)</td>
</tr>
<tr>
<td><strong>ICU</strong></td>
<td>hardware IRQ controller / manager</td>
</tr>
<tr>
<td><strong>SCHEDULER</strong></td>
<td>manage CPUs and scheduling</td>
</tr>
</tbody>
</table>
TASK

(class Task : Kobject)

PASSIVE PROTECTION DOMAIN (NO THREADS)
Memory protection (incl. IA32 IO-ports)
- virtual memory space (MMU)
- kernel-user memory (KU memory)

Access control (kernel objects, IPC)
- object space (capability table)
VIRTUAL MEMORY MODEL

Linux App

Native L4 App / Driver

L4Linux Kernel

Resource Management

RAM

Device Mem

MICROKERNEL CONSTRUCTION
VIRTUAL MEMORY MODEL II

COMPLETELY USER MANAGED
- no sbrk ...

GRANT ACCESS FROM A TO B
- directly, virtual addr. in A to virtual addr. in B
- using IPC (MapItem, Flexpage)

PAGE-FAULT HANDLING ON USER LEVEL
- translate page faults to messages
OBJECT SPACE

CONTAINS CAPABILITIES TO KERNEL OBJECTS
SIMILAR TO VIRTUAL MEMORY...

- completely user-level managed
- pass capabilities directly or via IPC (MapItem, Flexpage)
EXECUTES IN A TASK
access to virtual memory and capabilities of that task

STATES: ready, running, blocked

UTCB: for message contents (sys-call parameters)

ACTIVE ENDPOINT IN SYNCHRONOUS IPC

NEED SCHEDULER OBJECT TO SETUP SCHEDULING

EXTENDED FEATURES: vCPU mode (later)
FACTORY

CREATE (KERNEL) OBJECTS
- limited by kernel-memory quota
- secondary kernel memory also accounted page tables, KU memory, FPU state buffers, mapping nodes

GENERIC INTERFACE
- used for kernel and user-level objects
IPC GATE

COMMUNICATION CHANNEL
- messages forwarded to a thread
- incl. protected label for identification
- fundamental primitive for user-level objects

(class Ipc_gate : Kobject)
ASYNCHRONOUS SIGNALING

- signal forwarded as message to thread
- no payload
- incl. protected label for identification
- fundamental primitive for hardware IRQs and software signaling

(class Irq : Kobject)
INTERRUPT CONTROLLER ABSTRACTION
- bind IRQ object to a hw IRQ pin / source
  IRQ object gets triggered by hardware interrupt
- control parameters of IRQ pin / source

GENERIC INTERFACE
- used also for virtual IRQ sources
SCHEDULER

(class Scheduler : Kobject)

MANAGE CPUs and CPU TIME
- bind thread to CPU
- control scheduling parameters
- gather statistics

GENERIC INTERFACE
- used to refine resource management policies
KERNEL INTERFACE

Operation

UTCB

MRs

BRs

TCRs

LABEL

TAG

p
i
w
f

capability

Object

Message

Opcode Parameters

Thread

Microkernel Construction
MESSAGES

**TAG** message descriptor
  - number of words
  - number of items
  - flags
  - protocol id (payload)

**LABEL** protected message payload
  - secure identification of a specific capability a message was sent through
MESSAGES: UTCB user-level thread control block

**MR**  
- message registers  
  - untyped message data  
  - message items (capabilities, memory pages, IO ports)

**BR**  
- buffer registers  
  - receive buffers for capabilities, memory, IO-ports  
  - absolute timeouts

**TCR**  
- thread control registers  
  - error code  
  - user values
COMMUNICATION (IPC)

Sender Thread

Sender Thread

capability

IPC GATE

LABEL

Sender Thread

Sender Thread

BRs

BRs

TCRs

TCRs

MRs

MRs

LABEL

TAG

p i w f

p i w f

UTCB

UTCB

Receiver Thread

Receiver Thread

LABEL

TAG

p i w f

p i w f

长远Kernel Construction
SYNCHRONOUS
- sender waits until the receiver is ready to receive
- blocking can be limited by a timeout

ATOMIC DATA ONLY IPC
- map IPC is not atomic (may block in map)

ATOMIC SEND–RECEIVE TRANSITION IN CALL
- reply can have a zero timeout
ASYNC. IRQ MESSAGE

Device

User App

HW IRQ
trigger sys-call

IRQ
LABEL

Receiver Thread

UTCB
MRs
BRs
TCRs

LABEL
TAG
p i w f
PAGE-FAULT MESSAGE

Faulting Thread

capability

Page-fault message

Fault Address ...

p i w f

IPC GATE

LABEL

UTCB

MRs

BRs

TCRs

LABEL TAG

p i w f

Receiver Thread
WHAT IS A VCPU?

THREAD MODEL
- Execute (x)or block+receive-mgs

CPU
- Execute and receive messages

VCPU
- Add asynchronous execution model to threads
A VCPU IS A THREAD
- every thread can be a vCPU

INTERRUPT-STYLE EXECUTION
- events (incoming IPCs and exceptions) transition the execution to a user-defined entry point
- virtual interrupt flag allows control

VIRTUAL USER MODE
- a vCPU thread can temporarily switch to a different task
VCPU STATE PAGE

ENTRY INFORMATION
- entry-point program counter
- entry stack pointer

VCPU STATE
- current mode
- exception, page-fault, and interrupt acceptance
STATE SAVE AREA
- entry-cause code
- complete CPU register state
- saved vCPU state, saved version of the vCPU state

IPC/IRQ RECEIVE STATE
- message parts usually delivered in CPU registers
## REAL CPU vs. VCPU

### COMPARISON OF REAL CPU AND VCPU

<table>
<thead>
<tr>
<th></th>
<th>Physical CPU</th>
<th>vCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency control</td>
<td>Interrupt flag</td>
<td>Virtual interrupt flag</td>
</tr>
<tr>
<td>Control flow transfer</td>
<td>Interrupt entry vector</td>
<td>Entry point</td>
</tr>
<tr>
<td>State recovery</td>
<td>Kernel stack by CPU</td>
<td>State-save area</td>
</tr>
<tr>
<td>MMU</td>
<td>Page-tables</td>
<td>Host tasks</td>
</tr>
<tr>
<td>Protection</td>
<td>Modes: Kernel / User</td>
<td>vCPU User / vCPU Kernel</td>
</tr>
</tbody>
</table>
USER PROCESSES? — NEED SEPARATE TASKS
- vCPU can migrate between tasks
HARDWARE ASSISTED VIRTUALIZATION

REQUIRES PRIVILEGED INSTRUCTIONS → HOST KERNEL IMPLEMENTED USING THE VCPU EXECUTION MODEL.

- State-save-area extended to hold
  - x86: VMCB/VMCS
  - ARM: PL1 state + vGIC state

- Guest memory for VM
  - x86: L4::VM, a specialized L4::Task
  - ARM: L4::Task
  - Mapping of guest physical memory
VCPUS AND FULL VIRTUALIZATION

VCPU EXTENDED CONTROL
VCPU-RESUME IMPLEMENTS VT/SVM FUNCTIONALITY
- Sanity checking on VMCS / VMCB

VMM CAN RUN WITH OPEN AND CLOSED VCPU IRQS:
- Open: VMM continues in entry upon VMexit
- Closed: VMM continues after resume upon Vmexit

NPT/EPT vs. vTLB
L4Re — L4 RUNTIME ENV

OVERVIEW
- component-based architecture

INTERFACES
- shared memory — data spaces

SERVICES

LIBRARIES

C vs C++
HIERARCHICAL SYSTEM
- layered security policies
- resource management
- trust management

OS FRAMEWORK
- application APIs
- guest OS APIs
UNIFORM API CONCEPTS
- same principles for microkernel and L4Re APIs

HIGHER-LEVEL ABSTRACTIONS
- new high-level interfaces (data space, region map...)

SERVICES...
LIBRARIES...
L4Re — INTERFACES

DATA SPACE (class L4Re::Data_space)
- abstract container for memory (RAM, files, device MMIO...)

REGION MAP (class L4Re::Rm)
- address-space management (virtual memory)

ALLOCATOR (class L4Re::Mem_alloc)
- RAM allocation
L4Re — SERVICES

MEMORY MANAGEMENT
- data spaces
- allocator interface
- region map

PROGRAM LOADING
- ELF loading
- initial resource and capability setup

IO DEVICE ABSTRACTION
L4Re — LIBRARIES

C LIBRARY (uClibc)
P-THREAD (derived from uClibc p-threads)
STD C++ (from GCC)
**L4Re — C vs C++**

**CORE LANGUAGE IS C++**
- gain robustness by type-rich programming
- interfaces modeled as classes
- capabilities as (smart) pointers

**C BINDINGS FOR MOST INTERFACES**
- OS rehosting
- application porting
DATA SPACE, RM, SH-MEM

REGION MAP (RM)
- one per task
- virtual-address to data-space mapping

USE CASE
- ELF binary read-only shared in A & B
- DATA sections copied (cow) writable for A & B
- writable shared memory (sh mem) in A & B
- MMIO device mapping in B