Microkernel Construction
Introduction

Nils Asmussenn

04/11/2024
The lectures will be in-person in APB/E008

The lectures will also be live-streamed via BBB:
https://bbb.tu-dresden.de/b/nil-idy-kbw-ocw

Exercises will only be in-person (room TDB)

Questions can be asked live or on the mailing list
More Organization

- Thursday, 4th DS, 2 SWS
- Slides:
  - www.tudos.org → Studies → Lectures → MKC
- Subscribe to our mailing list:
  - www.tudos.org/mailman/listinfo/mkc2024
- In winter term:
  - Microkernel-based operating systems (MOS)
  - Various labs
Goals

1. Provide deeper understanding of OS mechanisms
2. Look at the implementation details of microkernels
3. Make you become enthusiastic microkernel hackers
4. Propaganda for OS research done at TU Dresden and Barkhausen Institut
Organization

Monolithic vs. Microkernel
- Kernel design comparison
- Examples for microkernel-based systems
- Vision vs. Reality
- Challenges

Overview About L4/NOVA
System components run in privileged mode

- No protection between system components
  - Faulty driver can crash the whole system
  - Malicious app could exploit bug in faulty driver
  - More than 2/3 of today’s OS code are drivers
- No need for good system design
  - Direct access to data structures
  - Undocumented and frequently changing interfaces
- Big and inflexible
  - Difficult to replace system components
  - Difficult to understand and maintain

Why something different?

→ Increasingly difficult to manage growing OS complexity
Microkernel System Design

System Services
- File Systems
- Network Stacks
- Memory Management
- Process Management
- Drivers

Microkernel
- Tasks
- Threads
- IPC
- Sched

Hardware

Application

User Mode
Kernel Mode
Example: QNX on Neutrino

1. Commercial, targets embedded systems
2. Network transparency

- Application
- System Services
  - Filesystem Manager
  - Device Manager
  - Process Manager
  - Network Manager
- Neutrino Microkernel
  - IPC
  - Interrupt Redirect
  - Sched
  - Network Driver
- Hardware
Example: L4Re on Fiasco.OC

1. Developed at our chair, now at Kernkonzept
2. Belongs to the L4 family

- Lx Application
- L4Re Application
- L4Linux
- Dope
- VPFS
- L4Re
- Fiasco.OC Microkernel
  - Task
  - Thread
  - IPC
  - IRQ
  - Sched
- Hardware
Genode is a spin-off of the chair

NOVA was built at our chair
Started at our chair, now continued at Barkhausen Institut

Similar to L4, but using a hardware/OS co-design
Vision vs. Reality

- **Flexibility and Customizability**
  - Monolithic kernels are typically modular

- **Maintainability and complexity**
  - Monolithic kernels have layered architecture

- ✔ Robustness / Security
  - Microkernels are superior due to isolated system components
  - Trusted code size
    - NOVA: 9,000 LOC
    - Linux: > 1,000,000 LOC (without drivers, arch, fs)

- ✗ Performance
  - Application performance degraded
  - Communication overhead (see next slides)
- Monolithic kernel: 2 kernel entries/exits
- Microkernel: 4 kernel entries/exits + 2 context switches
Monolithic kernel: 2 function calls/returns
Microkernel: 4 kernel entries/exits + 2 context switches
Challenges

1. Build functionally powerful and fast microkernels
   - Provide abstractions and mechanisms
   - Fast communication primitive (IPC)
   - Fast context switches and kernel entries/exits

   → Subject of this lecture

2. Build efficient OS services
   - Memory management
   - Synchronization
   - Device drivers
   - File systems
   - Communication interfaces

   → Subject of lecture ”Microkernel-based operating systems”
Outline

- Organization
- Monolithic vs. Microkernel
- Overview About L4/NOVA
  - Introduction
  - Kernel Objects
  - Capabilities
  - IPC
L4 Microkernel Family

Originally developed by Jochen Liedtke (GMD / IBM Research)

Current development:
- UNSW/OKLABS: OKL4, seL4
- TU Dresden/Kernkonzept: Fiasco.OC
- Bedrock Systems/Genode Labs/Cyberus Technology: NOVA
- Barkhausen Institut: M³
More Microkernels (incomplete)

- Singularity @ Microsoft Research
- K42 @ IBM Research
- Chorus/ChorusOS @ Sun Microsystems
- PikeOS @ SYSGO AG
- EROS/CoyotOS @ John Hopkins University
- Minix @ FU Amsterdam
- Pistachio @ KIT
- Barrelfish @ ETH Zurich
- Harmony OS @ Huawei
- Fuchsia with Zircon microkernel @ Google
L4 Concepts

1. Jochen Liedtke: “A microkernel does no real work”
   - Kernel provides only inevitable mechanisms
   - No policies implemented in the kernel

2. Abstractions
   - Tasks with address spaces
   - Threads executing programs/code

3. Mechanisms
   - Resource access control
   - Scheduling
   - Communication (IPC)
Why NOVA?

- NOVA is small and simple ($\approx 9000$ SLOC)
- NOVA is arguably elegant
- NOVA is efficient
- NOVA is open source:
  
  https://github.com/udosteinberg/NOVA
Why NOVA: TCB Size

Why NOVA: Performance (Linux kernel compilation)

Protection Domain (PD)

- PD is a resource container
  - Object capabilities (e.g., PD, execution context, …)
  - Memory capabilities (pages)
  - I/O port capabilities (NOVA runs only on x86)
- Capabilities can be exchanged between PDs
- Typically, PD contains one or more execution contexts
- Not hierarchical (in the kernel)

NOVA to Fiasco.OC

Protection Domain \(\sim\) Task
Execution Context (EC)

- EC is the entity that executes code
  - User code (application)
  - Kernel code (syscalls, pagefaults, IRQs, exceptions)
- Has a user thread control block (UTCB) for IPC
- Belongs to exactly one PD
- Receives time to execute from scheduling contexts
- Pinned on a CPU (not migratable)
- Three variants: Local EC, Global EC and VCPU

NOVA to Fiasco.OC

Execution Context + Scheduling Context $\sim$ Thread
Scheduling Context (SC)

- SC supplies an EC with time
- Has a budget and a priority
- NOVA schedules SCs in round robin fashion
- Scheduling an SC, activates the associated EC

NOVA to Fiasco.OC

Execution Context $+$ Scheduling Context $\sim$ Thread
A portal is an endpoint for synchronous IPC
Each portal belongs to exactly one (Local) EC
Calling a portal, transfers control to the associated EC
Data and capability exchange via UT CB
No cross-core IPC

NOVA to Fiasco.OC
Portal ≃ IPC Gate
- A semaphore offers asynchronous communication (one bit)
- Supports: up, down and zero
- Can be used cross-core
- Hardware interrupts are represented as semaphores

NOVA to Fiasco.OC

Semaphore ≃ IRQ
Capabilities

- Access to kernel objects is provided by capabilities
- Capability is a pair: (pointer to kernel object, permissions)
- Every PD has its own capability space (local, isolated)
- Capabilities can be exchanged:
  - Delegate: copy capability from one Cap Space to the other
  - Revoke: remove capability, recursively
- Applications use selectors to denote capabilities

**NOVA to Fiasco.OC**
Delegate = Map
Capabilities Overview

Selector  Capability

K. Object  Cap Space

PD A
02 4
EC SM PD EC
u s r k r n l rw- r-- r-x

PD B
0 3 4
rw- r-- r-x...

Selector Capability K. Object Cap Space

PD A
2 0 4
rw- r-- r-x...

PD B
4 0 1 3
rw- r-- r- x...

user

kernel

EC SM PD PT EC
Interprocess Communication

Sender

(1) send

Data

(2) copy

Receiver

Kernel

Buf

CPU

31 / 32
**Introduction**
- Threads and address spaces
- Kernel entry and exit
- Interprocess communication
- Capabilities
- Exercise 1: kernel entry, exit
- Exercise 2: Linkerscript, Multiboot, ELF
- Exercise 3: Thread switching
- Case study: L4Re
- Case study: M³
- Case study: Escape