Microkernel Construction
Introduction

Nils Asmussen

04/04/2019
Outline

- Introduction
  - Goals
  - Administration
- Monolithic vs. Microkernel
- Overview About L4/NOVA
 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 at TU Dresden
Thursday, 4th DS, 2 SWS

Slides:

www.tudos.org → Teaching → Microkernel Construction

Subscribe to our mailing list:

www.tudos.org/mailman/listinfo/mkc2019

In winter term:

- Microkernel-based operating systems (MOS)
- Various labs
Outline

• Introduction

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

• Overview About L4/NOVA
Monolithic Kernel OS (Propaganda)

- **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

System Services
- Tasks
- Threads
- IPC
- Sched

Microkernel

Hardware

Application

Application

Application
Example: QNX on Neutrino

1. Commercial, targets embedded systems
2. Network transparency

![Diagram of Neutrino Microkernel and System Services]

- Application
- Application
- 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
Genode is a spin-off of the chair

NOVA was built at our chair
1. Started at our chair, now continued at Barkhausen Institut
2. Similar to L4, but for heterogeneous hardware
Vision vs. Reality

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

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

- ✔ Robustness
  - 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)
Performance vs. Robustness (1)

- Monolithic kernel: 2 kernel entries/exits
- Microkernel: 4 kernel entries/exits + 2 context switches
Performance vs. Robustness (2)

- 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

- Introduction
- Monolithic vs. Microkernel
- **Overview About L4/NOVA**
  - Introduction
  - Kernel Objects
  - Capabilities
  - IPC
1. Originally developed by Jochen Liedtke (GMD / IBM Research)
2. Current development:
   - UNSW/NICTA/OKLABS: OKL4, seL4
   - TU Dresden: Fiasco.OC, NOVA, (M$^3$)
3. Support for hardware architectures:
   - x86, ARM, …
More Microkernels (incomplete)

1. Commercial kernels:
   - Singularity @ Microsoft Research
   - K42 @ IBM Research
   - velOSity/INTEGRITY @ Green Hills Software
   - Chorus/ChorusOS @ Sun Microsystems
   - PikeOS @ SYSGO AG

2. Research kernels
   - EROS/CoyotOS @ John Hopkins University
   - Minix @ FU Amsterdam
   - Amoeba @ FU Amsterdam
   - Pebble @ Bell Labs
   - Grasshopper @ University of Sterling
   - Flux/Fluke @ University of Utah
   - Pistachio @ KIT
   - Barrelish @ ETH Zurich
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)
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 $\simeq$ 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 UTCB
No cross-core IPC

NOVA to Fiasco.OC
Portal ≃ IPC Gate
Semaphore (SM)

- 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
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
Interprocess Communication

Kernel

CPU

Sender Receiver
(1) send
(2) copy
EC
ECBuf Buf
Data

Kernel

CPU

Buf
EC

(2) copy

Kernel

CPU

Buf
EC
• **Introduction**
  - Threads and address spaces (April 18th)
  - Kernel entry and exit
  - Interprocess communication
  - Capabilities
  - Exercise: kernel entry, exit
  - Exercise: Linkerscript, Multiboot, ELF
  - Exercise: Thread switching
  - Case study: M³
  - Case study: Fiasco.OC?
  - Case study: Escape