Microkernel-based Operating Systems
—Introduction—

Jan Bierbaum, Carsten Weinhold

Dresden, October 10th 2023
Lecture Goals

• Provide deeper understanding of OS mechanisms
• Illustrate alternative design concepts
• Promote OS research at TU Dresden
• Make you all enthusiastic about OS development in general and microkernels in particular
• **Slides:** [https://tud.de/inf/os → Studies → Lectures → MOS](https://tud.de/inf/os)

• **Mailing list (announcement & updates):**
  [https://os.inf.tu-dresden.de/mailman/listinfo/mos2023](https://os.inf.tu-dresden.de/mailman/listinfo/mos2023)

• **Matrix:** [https://matrix.to/#/#inf-os-mos:tu-dresden.de](https://matrix.to/#/#inf-os-mos:tu-dresden.de)

• **Schedule:**
  - Lecture: every Tuesday, 16:40, APB/E023
  - Exercises: (roughly) bi-weekly, Tuesday, 14:50
  - (mostly) hybrid via BBB
  - **Switch slots of lecture and exercise?**
Exercises

• Practical exercises in the computer lab
• Paper reading exercises in APB/E008 (?)
  – Read a paper **beforehand**
  – Sum it up and prepare questions/observations
  – Actively participate in the discussion
• Exercises may also take place online
• Announced on website and mailing list
More Hands-On: Complex Lab

- Complex lab “Microkernel-Based Operating Systems” in parallel to this lecture
- Build several components of an MOS
- Online via BBB; time to be changed
- Mainly programming on your own
- Coordination via dedicated mailing list, check the complex lab website
Monoliths vs. Microkernels
Purpose of an Operating System

• Manage the available resources
  – Hardware (CPU, memory, ...)
  – Software (file systems, networking stack, ...)

• Provide easier-to-use interface to access resources
  – Unix: read/write data from/to sockets instead of fiddling with TCP/IP packets on your own

• Perform privileged / HW-specific operations
  – x86: ring 0 vs. ring 3
  – Device drivers

• Provide separation and collaboration
  – Isolate users / processes from each other
  – Allow cooperation if needed (e.g., sending messages between processes)
What's the Problem with Monoliths?

- **Security issues**
  - All components in privileged mode
  - Direct access to all kernel-level data
  - Module loading → easy living for rootkits

- **Resilience issues**
  - Faulty drivers can crash the whole system
  - 75% of today's OS kernels are drivers

- **Software-level issues**
  - Complexity is hard to manage
  - Custom OS for hardware with scarce resources?
The Microkernel Vision

- Minimal OS kernel
  - less error prone
  - small *Trusted Computing Base*
  - suitable for verification

- System services in user-level *servers*
  - flexible and extensible

- Protection between individual components
  - More resilient: crashing component does not (necessarily...) crash the whole system
  - More secure: inter-component protection
The Microkernel Vision

User mode
- Application
- Application
- Application
- Application

File Systems, VFS, File System Impl.
Networking, Sockets, Protocols
Memory Management, Page Allocation, Swapping
Device Drivers

Kernel mode
- System-Call Interface
- Address Spaces, Threads, Scheduling, IPC
- Hardware Access

Microkernel

Hardware
CPU, Memory, PCI, Devices
What Microkernels Can Give Us…

- OS personalities
- Customisability
  - Servers may be configured to suit the target system (small embedded systems, desktop PCs, SMP systems, ...)
  - Remove unnecessary servers
- Enforce reasonable system design
  - Well-defined interfaces between components
  - Access to components only via these interfaces
  - Improved maintainability
Mach: A 1st-generation Microkernel

- Developed at CMU, 1985 – 1994
  - Rick Rashid (former head of MS Research)
  - Avie Tevanian (former Apple CTO)
  - Brian Bershad (professor @ U. of Washington)
  - ...

- Foundation for several real systems
  - Single Server Unix (BSD4.3 on Mach)
  - MkLinux (OSF, Apple)
  - IBM Workplace OS
  - NeXT OS → Mac OS X
Mach: Features

• Simple, extensible *communication kernel*
  – “Everything is a pipe.”
  – *Ports* as secure communication channels
• Multiprocessor support
• Message passing by mapping
• Multi-server OS personality
• POSIX compatibility
• Shortcomings
  – Performance
  – Drivers in the kernel
Case Study: IBM Workplace OS

- **Main goals:**
  - Multiple OS personalities
  - Support for multiple HW architectures

![Diagram showing OS personalities and services](image)

- Windows Personality
- Unix Personality
- OS/2 Personality

- Win Apps
- Unix Apps
- OS/2 Apps

- Network
- Processes
- Power
- Files
- OS base services

- Mach microkernel

- ARM
- PPC
- x86
- MIPS
- Alpha
IBM Workplace OS: A Failure?

• Never finished ... but almost 2 billion US-$ spent
• Causes of failure:
  – Underestimated difficulties in creating OS personalities
  – Management errors: divisions forced to adopt new system without having a system
  – "Second System Effect": too many fancy features
  – Too slow
• Conclusion: Microkernel worked, but system atop the microkernel did not
IBM Workplace OS: Lessons Learned

- OS personalities did not work
- Flexibility ... but monolithic kernels became flexible, too (e.g., Linux kernel modules)
- Better design ... but monolithic kernels also improved (restricted symbol access, layered architectures)
- Maintainability ... still very complex
- Performance matters a lot
Microkernels: Proven Advantages

• **Subsystem protection / isolation**

• **Code size** (generated using David A. Wheeler's 'SLOCCount')
  - Microkernel-based OS
    - Fiasco kernel: ~34,000 LoC
    - “HelloWorld” (+ boot loader + root task): ~10,000 LoC
  - Linux kernel (3.0.4., x86 architecture):
    - Kernel: ~2.5 million LoC
    - + Drivers: ~5.4 million LoC

• **Customisability**
  - Tailored memory management / scheduling / … algorithms
  - Adaptable to embedded / real-time / secure / … systems
Challenges

- We need fast and efficient kernels
  - Covered in the “Microkernel construction” lecture in the summer term
- We need fast and efficient OS services
  - Memory and resource management
  - Synchronization
  - Device Drivers
  - File systems
  - Communication interfaces
  → Subject of this lecture
Who's Out There?

- Minix @ FU Amsterdam (Andrew Tanenbaum)
- Singularity @ MS Research
- EROS/CoyotOS @ Johns Hopkins University
- The L4 Microkernel Family
  - Originally developed by Jochen Liedtke at IBM and GMD
  - 2nd-generation microkernel
  - Several kernel ABI versions
The L4 Family Tree

University of New South Wales / NICTA / Open Kernel Labs

University of Karlsruhe

TU Dresden

University of Karlsruhe

TU Dresden

ABI Specification
Implementation

seL4
OKL4v2
OKL4

NICTA::Pistachio-embedded

L4/x86
L4Ka::Hazelnut
L4Ka::Pistachio

Fiasco/L4v2
L4.Sec
Fiasco/L4.Fiasco
Fiasco
OC
Fiasco.OC

v2
x0
x2/v4

L2, L3

TU Dresden, 2023-10-10
MOS — Introduction
• Jochen Liedtke:
  “A microkernel does no real work.”
  – Kernel only provides inevitable mechanisms.
  – Kernel does not enforce policies.

• But what is inevitable?
  – Abstractions
    • Threads
    • Address spaces (tasks)
  – Mechanisms
    • Communication
    • Resource mapping
    • (Scheduling)
Taking a closer look at L4

Case study: L4/Fiasco.OC
• “Everything is an object”
  - Task Address spaces
  - Thread Activities, scheduling
  - IPC Gate Communication, resource mapping
  - IRQ Communication
  - Factory Create other objects, enforce resource quotas

• One system call: `invoke_object()`
  - Parameters passed in UTCB
  - Types of parameters depend on type of object
• Kernel-provided objects
  – Threads
  – Tasks
  – IRQs
  – ...

• Generic communication object: IPC gate
  – Send message from sender to receiver
  – Allows to implement new objects in user-level applications
• Build everything above kernel using user-level objects that provide a service
  – Networking stack
  – File system
  – ...

• Kernel provides
  – Object creation/management
  – Object interaction: Inter-Process Communication (IPC)
To call an object, we need an address:
- Telephone number
- Postal address
- IP address
- ...

Simple idea, isn’t it?
ID is wrong? Kernel returns ENOTEXIST
But not so fast! This scheme is insecure:
- Client could simply “guess” IDs brute-force
- (Non-)Existence can be used as a covert channel
• Global object IDs are
  - insecure (forgery, covert channels)
  - inconvenient (programmer needs to know about partitioning in advance)

• Solution in Fiasco.OC
  - Task-local *capability space* as indirection
  - *Object capability* **required** to invoke object
  - Per-task name space
    • Maps names to object capabilities
    • Configured by task's creator
• Capability:
  – Reference to an object
  – Protected by the kernel
    • Kernel knows all capability–object mappings
    • Managed as a per-process capability table
    • User processes only use indices into this table

Client

**IPC Gate:** communication channel for “Service 1”

Kernel

invoke(capability(3))

Service 1
- Kernel object for communication: **IPC gate**

- Inter-process communication (IPC)
  - Between threads
  - Synchronous

- Sequence:
  - Sender writes message into its UTCB
  - Sender invokes IPC gate → blocks sender until receiver ready (i.e., waits for message)
  - Kernel copies message to receiver thread's UTCB
  - Both continue, knowing that message has been transferred/received
Capabilities == Local Names

Address Space

Address Space

Address Space

Address Space

Address Space

Address Space

Address Space
More L4 concepts
L4/Fiasco.OC: Threads

- **Thread**
  - Unit of execution
  - Implemented as kernel object

- **Properties managed by the kernel**
  - Instruction Pointer (EIP)
  - Stack (ESP)
  - Registers
  - User-level thread control block (UTCB)

- **User-level applications need to**
  - allocate stack memory
  - provide memory for application binary
  - find entry point
  - ...
• Kernel object: IRQ
• Used for hardware and software interrupts
• Provides asynchronous signaling
  - `invoke_object(irq_cap, WAIT)`
  - `invoke_object(irq_cap, TRIGGER)`
Problem: Memory partitioning

Physical Memory

App1

App2

App3

4 GB
Solution: Virtual Memory

- App1
- App2
- App3

Physical Memory

4 GB
L4: Recursive Address Spaces
L4: Resource Mappings

- If a thread has access to a capability, it can map this capability to another thread.
- Mapping / not mapping of capabilities used to implement access control.
- Abstraction for mapping: flexpage
  - Location and size of resource
  - Receiver's rights (read-only, mappable)
  - Type (memory, I/O, communication capability)
L4/Fiasco.OC: Object Types (Recap)

- Summary of object types
  - Task
  - Thread
  - IPC Gate
  - IRQ
  - Factory

- Each task gets initial set of capabilities for some of these objects at startup
What can we build with this?
• Fiasco.OC is **not** a full operating system!
  - No device drivers (except UART + timer)
  - No file system / network stack / …

• A microkernel-based OS needs to add these services as user-level components

L4Re = L4 Runtime Environment
L4Linux: Linux on L4

Linux Kernel:
- Networking, Sockets, Protocols
- Processes, Scheduling, IPC
- Memory Management, Page allocation, Address spaces, Swapping
- Device Drivers
- Hardware Access

System-Call Interface

Runtime Environment (L4Re)

User mode

Kernel mode

Application
L4 Task

Fiasco.OC

Hardware
The Dresden Real-Time Operating System

L^4 Linux

Non-RT World

RT World

Resource Management Layer (L4Re)

Fiasco.OC microkernel

User Mode

Kernel Mode

Apps

Time service

RT Apps

Network driver

SCSI driver

Display driver
• Isolate not only processes, but also complete operating systems (compartments)
• “Server consolidation”
- **Genode = C++-based OS framework developed here in Dresden**
  - Goal: hierarchical system that
    - supports resource partitioning
    - layers security policies on top of each other
What’s to come?
• **Basic mechanisms and concepts**
  - Memory management
  - Tasks, Threads, Synchronisation
  - Communication

• **Building real systems**
  - What are resources and how to manage them?
  - How to build a secure system?
  - How to build a real-time system?
  - How to re-use existing code (Linux, standard system libraries, device drivers, ...)?
  - How to improve robustness and safety?
Outlook

• Next lecture:
  – “Inter-Process Communication (IPC)”
  – Next week (Oct 17, 16:40)

• First exercise:
  – Per Brinch Hansen: “The nucleus of a multiprogramming system”
  – Link will be on the website
  – Oct 24, 14:50 (? → check the website/mailing list)
  – Read the paper!