Microkernel-based Operating Systems - Introduction

Carsten Weinhold

Dresden, Oct 13 2015
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 special
Organization: Lecture

- Lecture every Tuesday, 4:40 PM, INF/E01
- Slides: [http://www.tudos.org](http://www.tudos.org) -> Teaching -> Microkernel-based Operating Systems
- This lecture is **not**: Microkernel construction (in summer term)
Organization: Exercises

- Exercises (roughly) bi-weekly
  Tuesday, 2:50 PM, INF/E01
- Practical exercises in the computer lab
- Paper reading exercises
  - Read a paper beforehand.
  - Sum it up and prepare 3 questions.
  - We expect you to actively participate in discussion.
- First exercise: next week
  - Brinch-Hansen: *Nucleus of a multiprogramming system*
More Practical Stuff: Complex lab

• Complex lab in parallel to lecture
• Build several components of an OS
• “Komplexpraktikum” for (Media) Computer Science students
• Starts in 2 weeks, 2:50 PM, INF/E08
<table>
<thead>
<tr>
<th>Date</th>
<th>Lecture</th>
<th>Exercise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 13</td>
<td>Intro</td>
<td></td>
</tr>
<tr>
<td>Oct 20</td>
<td>Threads &amp; Synchronization</td>
<td>Paper: Nucleus of an MP System</td>
</tr>
<tr>
<td>Oct 27</td>
<td>IPC</td>
<td>(Complex Lab)</td>
</tr>
<tr>
<td>Nov 3</td>
<td>Memory Management</td>
<td>Practical: Booting Fiasco/L4Re</td>
</tr>
<tr>
<td>Nov 10</td>
<td>Real-Time</td>
<td></td>
</tr>
<tr>
<td>Nov 17</td>
<td>Device Drivers</td>
<td>Paper: Singularity</td>
</tr>
<tr>
<td>Nov 24</td>
<td>Resource Management</td>
<td>(Complex Lab)</td>
</tr>
<tr>
<td>Dec 1</td>
<td>Practical: IPC</td>
<td>Heterogeneous Systems</td>
</tr>
<tr>
<td>Dec 8</td>
<td>Legacy Containers</td>
<td>(Complex Lab)</td>
</tr>
<tr>
<td>Dec 15</td>
<td>Virtualization</td>
<td></td>
</tr>
<tr>
<td>Jan 5</td>
<td>Secure Systems</td>
<td>(Complex Lab)</td>
</tr>
<tr>
<td>Jan 12</td>
<td>L4/Android</td>
<td>Practical: L4Linux</td>
</tr>
<tr>
<td>Jan 19</td>
<td>Trusted Computing</td>
<td>Paper: Survey of VM Research</td>
</tr>
<tr>
<td>Jan 26</td>
<td>Faults, Failures &amp; Resilience</td>
<td>Paper: Microkernels meet ...</td>
</tr>
<tr>
<td>Feb 2</td>
<td>Genode ???</td>
<td>(Complex Lab)</td>
</tr>
</tbody>
</table>
Purpose of Operating Systems

• 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)
Monolithic kernels: Linux

- User mode
  - Application
  - Application
  - Application
  - Application

- Kernel mode
  - Linux Kernel
    - System-Call Interface
      - File Systems
        - VFS
      - Networking
        - Sockets
      - Processes
        - Scheduling
      - IPC
      - Memory
        - Management
        - Page allocation
        - Address spaces
        - Swapping
    - Device Drivers
    - Hardware Access

Hardware
- CPU, Memory, PCI, Devices
What's the problem?

- **Security issues**
  - All components run 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
  - Hardware
    - CPU, Memory, PCI, Devices
What microkernels can give us ...

- OS personalities
- Customizability
  - Servers may be configured to suit the target system (small embedded systems, desktop PCs, SMP systems, ...)
  - Remove unneeded servers
- Enforce reasonable system design
  - Well-defined interfaces between components
  - No access to components besides these interfaces
  - Improved maintainability
The mother of all microkernels

• Mach – developed at CMU, 1985 - 1994
  – Rick Rashid (today 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)
  – IBM Workplace OS
  – NeXT OS → Mac OS X
• 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 still in the kernel
**Main goals:**
- multiple OS personalities
- run on multiple HW architectures

- Win Apps
  - Windows Personality
- Unix Apps
  - Unix Personality
- OS/2 Apps
  - OS/2 Personality

OS base services:
- Network
- Processes
- Power
- Files
- ...
IBM Workplace OS: Why did it fail?

- Never finished (but spent 1 billion $)
- Failure causes:
  - Underestimated difficulties in creating OS personalities
  - Management errors, forced divisions 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
• OS personalities did not work
• Flexibility – but monolithic kernels became flexible, too (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
  - 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
      - (generated using David A. Wheeler's 'SLOCCount')
- Customizability
  - 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 is (or was) 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 – a timeline (or tree ...)

University of New South Wales / NICTA / Open Kernel Labs

L2, L3 → v2 → x0 → x2/v4

L4/x86
L4Ka::Hazelnut
L4Ka::Pistachio

NICTA:: Pistachio-embedded
OKL4v2
OKL4

University of Karlsruhe

Fiasco/L4v2

L4Sec
Fiasco/L4.Fiasco
Fiasco.OC

TU Dresden

SeL4
ABI Specification
Implementation

Fiasco

Nova

TU Dresden, 2015-10-13
MOS - Introduction
Slide 21 von 47
L4 concepts

- 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**
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
L4/Fiasco.OC: Types of Objects

- Kernel-provided objects
  - Threads
  - Tasks
  - IRQs
  - ...

- Generic communication object: IPC gate
  - Send message from sender to receiver
  - Used to implement new objects in user-level applications
L4/Fiasco.OC: User-level objects

- Everything above kernel built 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, right?
ID is wrong? Kernel returns ENOTEXIST
But not so fast! This scheme is insecure:
- Client could simply “guess” IDs brute-force.
- Existence/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 an indirection
  - *Object capability* required to invoke object

• Per-task name space
  - Maps names to object capabilities.
  - Configured by task's creator
L4/Fiasco.OC: Object capabilities

- **Capability:**
  - Reference to an object
  - Protected by the Fiasco.OC kernel
    - Kernel knows all capability-object mappings.
    - Managed as a per-process capability table.
    - User processes only use indexes into this table.

```plaintext
Client \[1\] \[2\] \[3\] \[4\]

IPC Gate: communication channel for "Service 1"

Service 1

Kernel

invoke(capability(3))
```
• Kernel object for communication: **IPC gate**

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

• Communication using IPC gate:
  – Sender thread puts 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
Indirection allows for security and flexibility.
More L4 concepts
• **Thread**
  - Unit of Execution
  - Implemented as kernel object

• **Properties managed by the kernel:**
  - Instruction Pointer (EIP)
  - Stack (ESP)
  - Registers
  - User-level TCB

• **User-level applications need to**
  - allocate stack memory
  - provide memory for application binary
  - find entry point
  - ...

![Diagram of address space with threads, code, data, UTCBs, and stack sections]
• 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 for implementing access control

• Abstraction for mapping: flexpage

• Flexpages describe mapping
  - location and size of resource
  - receiver's rights (read-only, mappable)
  - type (memory, I/O, communication capability)
Summary of object types
- Task
- Thread
- IPC Gate
- IRQ
- Factory

Each task gets initial set of capabilities for some of these objects at startup
Building microkernel-based systems

What can we build with all 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
The Dresden Real-Time Operating System

Fiasco.OC microkernel

Resource Management Layer (L4Re)

Non-RT World

User Mode

Privileged Mode

RT World

Apps

L4Linux

Time service

Network driver

SCSI driver

Display driver

RT Apps

Non-RT World

RT World
Virtual machines

- Isolate not only processes, but also complete Operating Systems (compartments)
- “Server consolidation”

![Diagram showing virtualization layers]

- L4Linux
- L4Linux
- Native Linux
- VMM

User Mode

Virtualization Layer (L4Re)

Privileged Mode

Fiasco.OC microkernel
Genode

- Genode := C++-based OS framework developed here in Dresden
- Aim: hierarchical system in order to
  - Support resource partitioning
  - Layer security policies on top of each other
Lecture outline

- **Basic mechanisms and concepts**
  - Memory management
  - Tasks, Threads, Synchronization
  - 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 reuse existing code (Linux, standard system libraries, device drivers)?
  - How to improve robustness and safety?
Outlook

• Next lecture:
  - “Threads & Synchronization”
  - Next week (Oct 20, 4:40 PM)

• First exercise:
  - Per Brinch-Hansen: *The nucleus of a multiprogramming system*
  - Next week (Oct 20, 2:50 PM)
  - Read the paper! Link is on website!