Microkernel-based Operating Systems - Introduction

Björn Döbel

Dresden, Oct 14th 2008
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
Lecture every Tuesday, 1:00 PM, INF/E08
- Lecturers: Carsten Weinhold, Michael Roitzsch, Stefan Kalkowski, Björn Döbel

Slides: http://www.tudos.org -> Teaching -> Microkernel-based Operating Systems

Subscribe to our mailing list:

This lecture is not: Microkernel construction (in summer term)
• Exercises bi-weekly, Tuesday, 2:50 PM, INF/E08
• Practical exercises in the computer pool
• 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, computer pool
  - You'll need a quota raise.
• Complex lab in parallel to lecture
• Groups of 2-3 students.
• Build several components of an OS (memory server, keyboard driver, binary loader, ...)
• “Komplexpraktikum” for (Media) Computer Science students
• “Internship” for Computational Engineering
• starts on Tuesday, Oct 14\textsuperscript{th}
Monolithic kernels - Linux

User mode

Kernel mode

Linux Kernel

System-Call Interface

- File Systems
  - VFS
  - File System Impl.

- Networking
  - Sockets
  - Protocols

- Processes
  - Scheduling
  - IPC

- Memory
  - Management
  - Page allocation
  - Address spaces
  - Swapping

Device Drivers

Hardware Access

Hardware

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

- All system components run in privileged mode.
- No isolation of components possible.
  - Faulty driver crashes the whole system.
  - More than 2/3 of today's systems are drivers.
- No enforcement of good system design
  - Can directly access all kernel data structures
- Size and inflexibility
  - Not suitable for embedded systems.
  - Difficult to replace single components.
- Increasing complexity becomes more and more difficult to manage.
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
- Hardware Access
- Address Spaces
  - Threads
  - Scheduling
  - IPC

Microkernel

Hardware
- CPU, Memory, PCI, Devices
One vision - microkernels

- Minimal OS kernel
  - less error prone
  - small Trusted Computing Base
  - suitable for verification
- System services implemented as user-level servers
  - flexible and extensible
- Protection between individual components
  - systems get
    - More secure – inter-component protection
    - Safer – crashing component does not (necessarily...) crash the whole system
• Servers may implement multiple OS personalities
• Servers may be configured to suit the target system (small embedded systems, desktop PCs, SMP systems, ...)
• Enforce reasonable system design
  - Well-defined interfaces between components
  - No access to components besides these interfaces
  - Improved maintainability
Examples

- QNX kernel only contains
  - IPC
  - Scheduling
  - IRQ redirection

- LynxOS
  - “separation kernel”
  - combine secure and real-time components
The mother of all microkernels

- Mach – developed at CMU
  - designed as simple, extensible “communication kernel”
  - “ports” for communication channels and memory objects
- Foundation for several real systems
  - Single Server Unix (BSD4.3 on Mach)
  - MkLinux (OSF)
  - IBM Workplace OS
  - Mac OS X
- Shortcomings
  - performance
  - drivers still in the kernel
Mac OS X

- Hardware
  - Mach
  - BSD
  - Drivers, I/O kit

- Kernel
  - User space

- App Environments
  - BSD
  - Cocoa
  - Carbon

- Quartz Window Manager

- Application services

- Core services

- Quick Time

- AWT, Swing

- JVM

- JRE

- Core services

- User space

- Hardware
IBM Workplace OS

• Main goals:
  – multiple OS personalities
  – run on multiple HW architectures
IBM Workplace OS (2)

- Never finished
- 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
Lessons learned

- 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
Proven advantages

- Subsystem protection / isolation
- Code size
  - Fiasco kernel: \(~ 15,000 \text{ LoC}\)
  - Minimal application:
    (boot loader + “hello world”):
    \(~ 6,000 \text{ LoC}\)
  - Linux kernel (2.6.24, x86 architecture):
    \(~ 1.6 \text{ million LoC}\)
    (+drivers: \(~ 2.8 \text{ million LoC}\))
  (generated using David A. Wheeler's 'SLOCCount')
- Customizable
  - 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 (Tanenbaum)
- Singularity @ MS Research
- Eros/CoyotOS @ Johns Hopkins University
- The L4 Microkernel Family
  - Originally developed by Jochen Liedtke at IBM and GMD
  - 2\textsuperscript{nd} generation microkernel
  - Several kernel ABI versions
The L4 family – a timeline

Univ. of New South Wales / NICTA / Open Kernel Labs

L2, L3 → v2 → x0 → x2/v4 → N1 → N2 → SeL4

NICTA:: Pistachio-embedded → OKL4v2

OKL4

L4/x86

Univ. of Karlsruhe

Fiasco/L4v2 → L4Ka::Hazelnut → L4Ka::Pistachio

TU Dresden

Fiasco/L4.Fiasco → L4.Sec → fiasco_dev

Fiasco

Fiasco.XY?

TU Dresden, 2008-10-14

MOS - Introduction

Slide 20 von 37
L4 concepts

- Jochen Liedtke: “A microkernel does no real work.”
  - kernel provides inevitable mechanisms
  - kernel does not enforce policies
- But what is inevitable?
  - Abstractions
    - Threads
    - Address spaces (tasks)
  - Mechanisms
    - Communication
    - Mapping
    - (Scheduling)
L4 - Threads

- Thread ::= Unit of Execution
- Unique Thread ID
- Properties managed by L4 kernel:
  - Instruction Pointer (EIP)
  - Stack (ESP)
  - Registers
- User-level applications need to
  - allocate stack memory
  - provide memory for application binary
  - find entry point
  - ...  
- 1 addr. space contains up to 128 threads
• Synchronous inter-process communication (IPC) between threads
  - agreement between partners necessary
  - timeouts
  - no in-kernel buffering
  - efficient implementation necessary

• IPC flavors:
  - send
  - receive_from (closed wait)
  - receive (open wait)
  - call (send and receive_from)
  - reply and wait (send and receive)
L4 IPC – Message types

- short (register-only) IPC
- fast – no memory access

Thread A

send(…)

Thread B

receive(…)

EBX  EDX

EBX  EDX
L4 IPC – Message types

- Direct long IPC – more than 2 words at a time
- Words are directly copied:

Thread A

send(msg, ...)

Thread B

receive(msg, ...)

copy
• Indirect Long IPC (String IPC)
• Words in message buffer point to external memory areas that are copied
• Threads can map pages from their address space to other address spaces.
• This is achieved by adding a Flexpage descriptor to the IPC message buffer.
• Flexpages describe mapping
  – location and size of memory area
  – receiver's rights (read-only, read-writable)
  – type (memory, IO, communication capability)
• More general: flexpages as fundamental resource abstraction
L4 – Recursive address spaces

[Diagram showing recursive address spaces with layers labeled RAM, Physical Memory, and Device Memory]
• Special Thread ID to receive HW interrupts from the kernel
• Exactly one thread can listen to exactly one interrupt – multiplexing in userspace necessary.
• I/O Memory and I/O ports are manages using flexpages.

```
int ipc_recv(int irq_id, ...) {
    // ... implementation ...
    return 0;  // or other return value
}
```
System Calls in L4.Fiasco

- **Address spaces**
  - `l4_task_new` - create/delete tasks

- **Threads**
  - `l4_thread_ex_regs` - create/modify threads
  - `l4_thread_schedule` - setup scheduling parameters
  - `l4_thread_switch` - switch to another thread

- **IPC**
  - `l4_ipc` - perform IPC
  - `l4_fpage_unmap` - revoke flexpage mapping
  - `l4_nchief` - find next chief
Linux on L4

Linux Kernel

- System-Call Interface
- File Systems
  - VFS
  - File System Impl.
- Networking
  - Sockets
  - Protocols
- Processes Scheduling
- IPC
- Memory Management
  - Page allocation
  - Address spaces
  - Swapping
- Device Drivers
- Hardware Access

User mode

L4 Task

L4 Environment

Kernel mode

Application

L4 Task

Application

L4 Task

Application

L4 Task

Application

L4 Task

Fiasco

Hardware

TU Dresden, 2008-10-14
MOS - Introduction
Slide 31 von 37
The Dresden Real-Time Operating System

- L4Linux
  - Resource Management Layer (L4Env)
  - Non-RT World
    - User Mode
      - Privileged Mode
    - L4.Fiasco microkernel
  - RT World
    - RT Apps
      - Network driver
      - SCSI driver
      - Display driver
    - Time service

- Apps
  - Privileged Mode
    - User Mode
Virtual machines

- Isolate not only processes, but also complete Operating Systems (compartments)
- “Server consolidation”
• Disadvantages of existing systems (microkernels as well as monoliths):
  - global naming
  - resource management and revocation difficult
  - hard to get security policies right
• Bastei := C++-based OS framework developed here in Dresden
  - recursive system design
  - capabilities
  - stacked security policies
Design alternatives

- **Hardware isolation**
  - x86 privilege rings, privileged instructions

- **Software isolation**
  - N. Wirth's Oberon language and OS since 1980s
  - Singularity from MS Research written in a dialect of C# (since 2000s)

- **Exokernels**
  - build OS interface into a library and link it to single applications (library OS)
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:
  - “Tasks, Threads and Synchronization” on Oct 21\textsuperscript{st}

• Next exercise:
  - Oct 21\textsuperscript{st}
  - Building and booting an L4 system