TECHNISCHE UNIVERSITÄT DRESDEN
Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Virtualization

Stefan Kalkowski

Dresden, 2007-12-11

So Far ...

- Basics
- Device Drivers
- Real-time
- Naming
- Resource Management

Today: Virtualization

- Introduction
- Motivation & classification, flavors
- NOVA – a µ-hypervisor
- L4Linux: Para-virtualization on top of L4
  - Architecture
  - Address space layout
  - Scenarios
  - Freezing

One possible definition ...

Virtualization is the creation of a virtual or logical (rather than actual) version of something, such as an operating system, a hardware platform, a storage device or network resources

Virtualization – a hype

- A lot of interest in the research community within the last years, e.g.:
  - SOSP 03: *Xen and the Art of Virtualization*
  - EuroSys 07: a whole session about virtualization
- Many new virtualization products:
  - QEmu, Virtualbox, Intel-VT, AMD-V
- Further increasing demand:
  - Vmware: from 240 to 4300 employees in the last 4 years

Virtualization - a new idea?

- Originates in IBM's CP/CMS series used on System/3xx mainframes (starting ~1964)
- **C**ontrol **P**rogram
  - hypervisor
- **C**ambridge **M**onitor **S**ystem
  - guest OS
- Virtual Memory
- **S**tart **I**nterpretative **E**xecution instruction
  - starting virtualization mode

Virtualization - Motivation

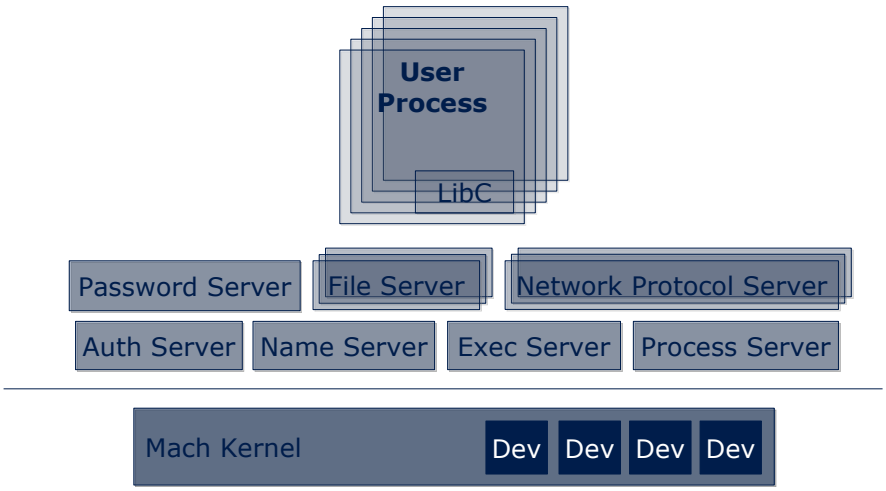
- Optimize utilization
- Maintainance
- Migration
  - reliability: hardware failure
  - load balancing
  - economic & ecological reasons: temporary shutdown of a system
- Isolation
- Support of different OS ABIs

Classification help

- Virtualization - an overloaded term
- Some classification criteria:
  - Objective target: hardware, OS API or ABI ?
  - Do we have to interpret all instructions ? : emulation vs. virtualization
  - Can we modify the target software ? : using para-virtualization techniques (L4Linux)
  - Is there any hardware support ? : hardware enabled virtualization (Intel-VT, AMD-V)

Virtualizing the operating system

- GNU Hurd example: rebuild the POSIX API



Emulation of OS interface

- Used to integrate a bunch of existing software to other respectively newly created OSes
- When emulating the API of an OS, target software needs to be re-linked
- In contrast to that, **ABI** emulation can run unmodified binaries e.g.: **W**ine **i**s **n**ot an **e**mulator
- Disadvantage of both approaches:
  - Great effort
  - Shooting at a moving target

Virtualizing the hardware

- Instead of emulating the OS API or ABI, take the underlying platform
- Full emulation:
  - QEMU can emulate of ARM processors for x86 platforms
- Virtualization of the underlying platform:
  - no additional interpretation of code running in user-mode
  - processor emulation only for kernel- and real-mode
  - Examples: KQEMU, Vmware, VirtualBox ...

Problems with x86 virtualization

- Ring-alias problem: guest OS cannot run in ring 0 (and can realize it)
- Some privileged instructions fail silently:
  - e.g.: `LAR`, `LSL`, `VERR`, `VERW`
- ... or behave unexpectedly:
  - e.g.: `SGDT`, `SIDT`, `SLDT`, `STR`
- Example: interrupt handling using POPF
- Faulting incessantly implies performance loss
  - kernel entry/exit -> doubled context switch

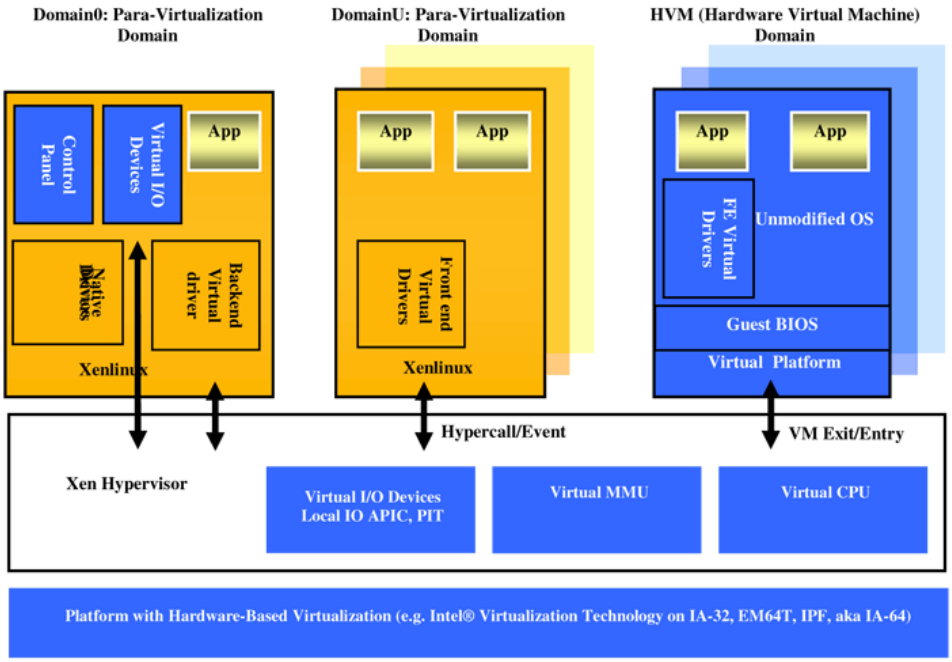
Platform virtualization in software

- Guest OS runs natively in less privileged mode
- Privileged instructions fail and are handled by the VMM (*trap-and-emulate*)
- VMM derives and manages *shadow* structures from guest's primary structures, e.g.: shadow page tables
- JIT binary translation
- Examples: VMWare, QEMU, VirtualBox

Hardware enabled virtualization

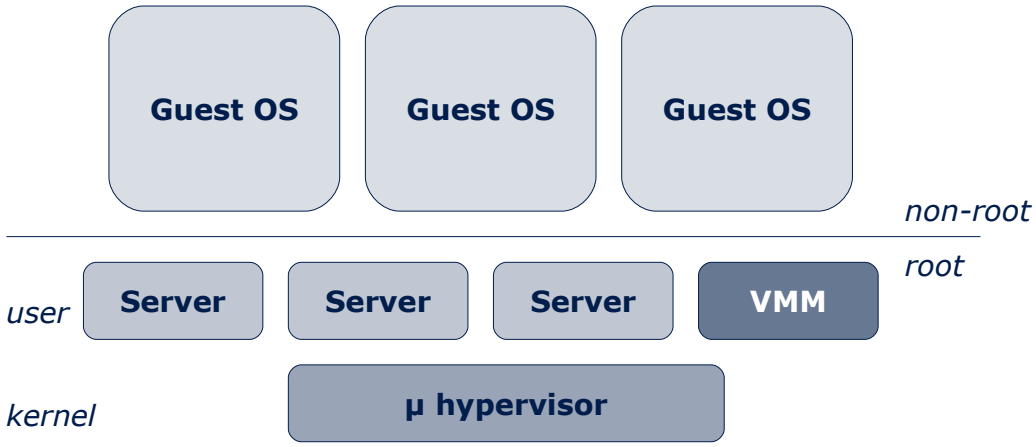
- Example Intel VT
- *root* and *non-root* mode, *VM entry* and *exit*
- **V**irtual **M**achine **C**ontrol **S**tructure in physical memory holds information of guest and host state and some additional control information
- VMCS is used to investigate *VM exit* conditions, e.g.: whether a specific Interrupt can be handled by the guest

XEN



NOVA – µ hypervisor approach

- **N**OVA **O**S **V**irtualization **A**rchitecture
- Separate hypervisor and VMM



NOVA

- Hypervisor manages protection domains:
  - address spaces and virtual machines
- Virtual machines have associated virtualization handlers -> the VMMs
- VMM handles virtualization faults and implements virtual devices
- Splitting functionality of hypervisor and VMM reduces the kernel size -> security relevant

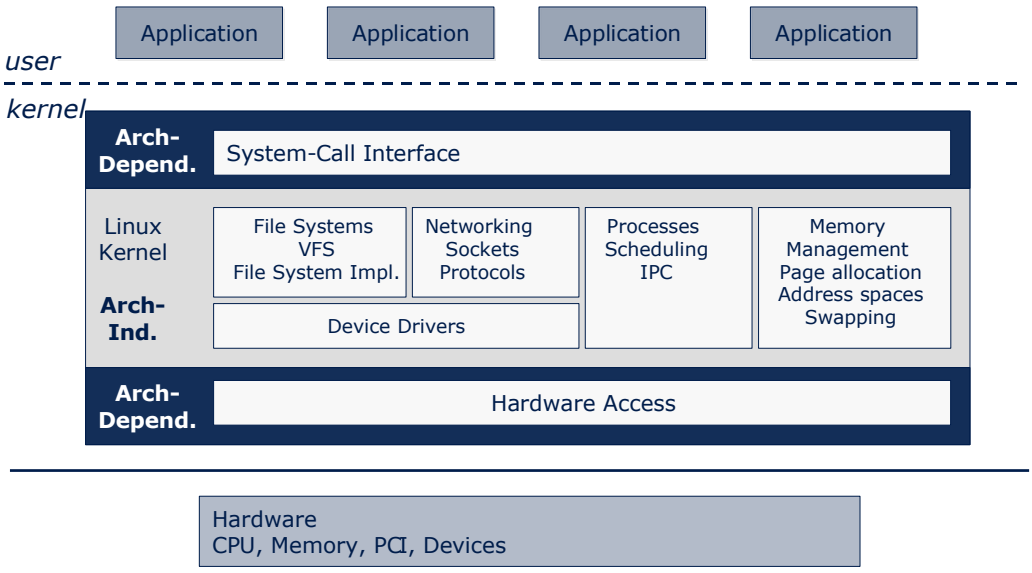
Paravirtualization

- Modify the guest OS source code to integrate it in the runtime environment
- Examples: L4Linux, Xen (without VT), UML
- Afterburner (Karlsruhe): modify binary code
- Advantages:
  - no hardware support necessary
  - effort negligible with respect to OS emulation
  - cooperation between guest OS (and its applications) and the native system

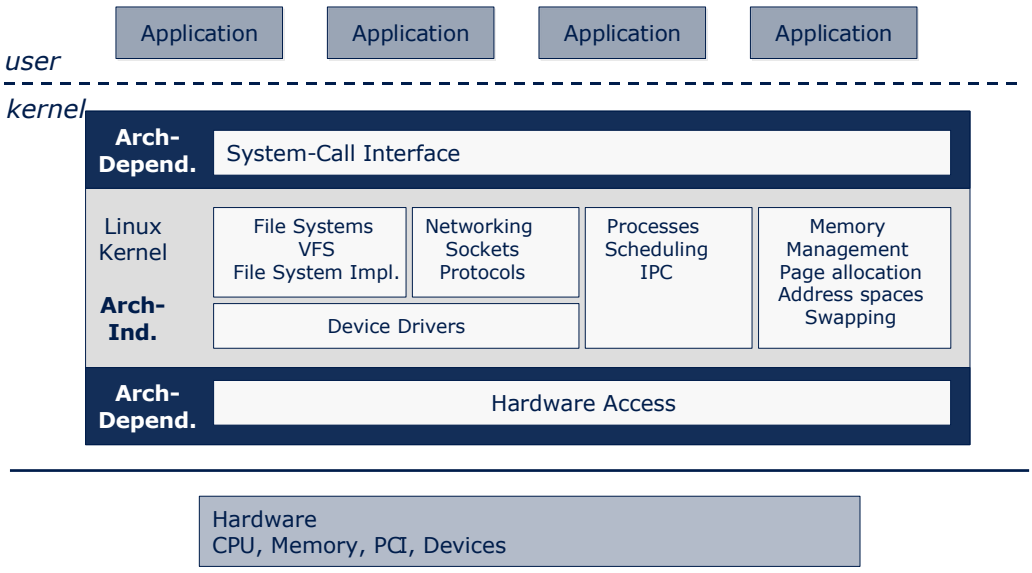
L4Linux: history

- Presented at SOSP '97
  - based on x86 Linux 2.0 on top of first L4 kernel
- (L4)Linux has evolved over the years
  - 2.2 supported MIPS and x86
  - 2.4 first version to run on L4Env
  - 2.6 uses 'paravirtualization' L4 kernel features
- Recently
  - Latest Linux release 2.6.23
  - ARM support
  - Freeze functionality
  - L4Env and Bastei support
  - SMP

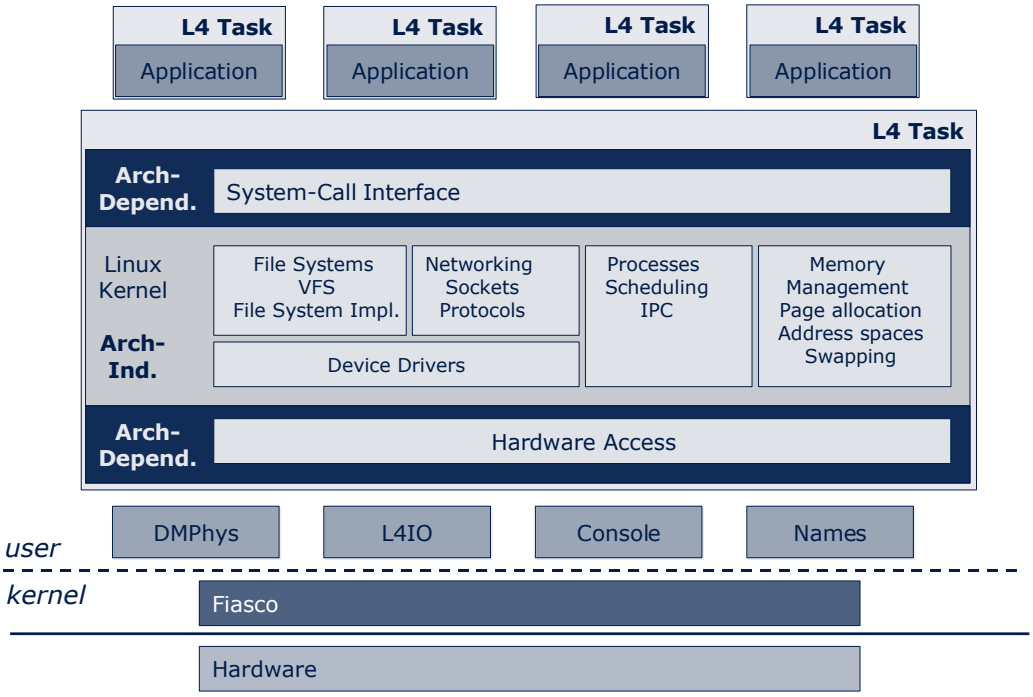
Linux Architecture



- Architecture dependent part
  - Small, for x86 about 2% of the kernel
  - System call interface:
    - Kernel entry
    - Signal delivery
    - Copy from/to user space
  - Hardware access:
    - CPU state and features
    - MMU
    - Interrupt
    - Memory mapped I/O, I/O ports
- Architecture dependent part implements generic interface used by independent part



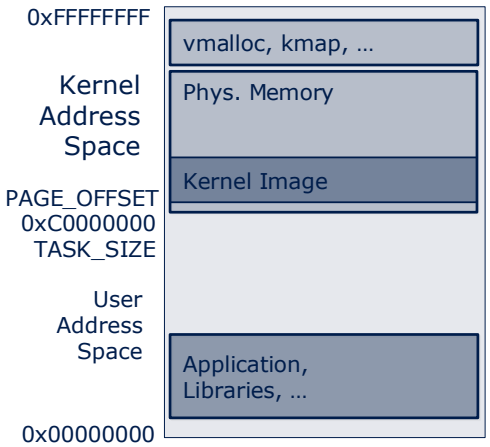
L4Linux Architecture



L4Linux Architecture

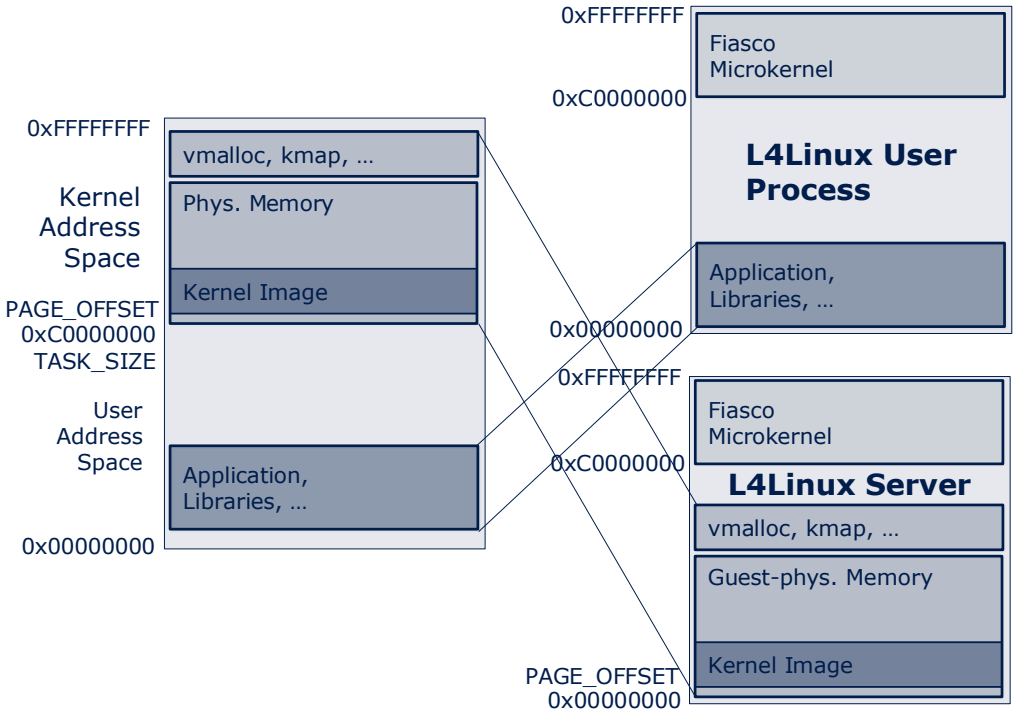
- Linux kernel and Linux user processes run each within a single L4 task
- L4/L4Env specific part is implemented as an own architecture: `arch/l4 include/asm-l4`
- L4/L4Env architecture dependent part itself divides into x86 and ARM specific part
- Most code is reused from x86 resp. ARM specific part

Linux address space layout



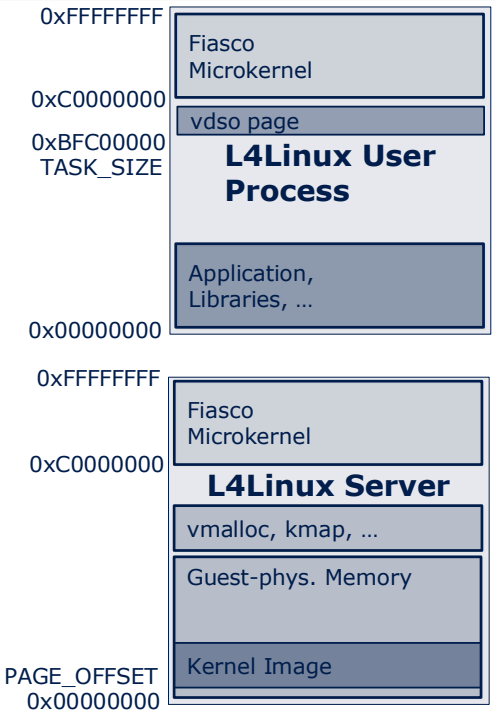
- `0x0 – TASK_SIZE`
  - user part
  - changes on every context switch
- `TASK_SIZE – 0xF...`
  - kernel part
  - constant in all address spaces
- Physical memory mapped beginning at `PAGE_OFFSET`

L4Linux address space layout



L4Linux address space layout

- L4Linux user task
  - little change
  - TASK_SIZE – VDSO
- VDSO page
  - originally in kernel part of address space
  - virtual dynamic shared object for 'vsyscall'
  - contains architecture dependent kernel entry code

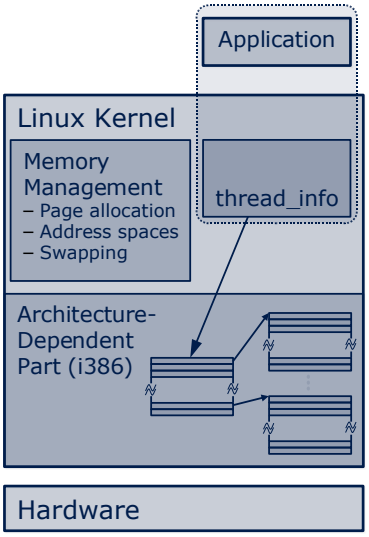


L4Linux: problems to be solved

- L4Linux server has to:
  - have some basic resources (memory, I/O)
  - manage page tables of its user processes
  - handle exceptions from user processes
  - schedule its tasks
- L4Linux user processes have to:
  - 'enter' the L4Linux kernel (now in a different address space)
- Kernel needs information from user processes formerly accessible in the same address space, e.g.: syscall arguments

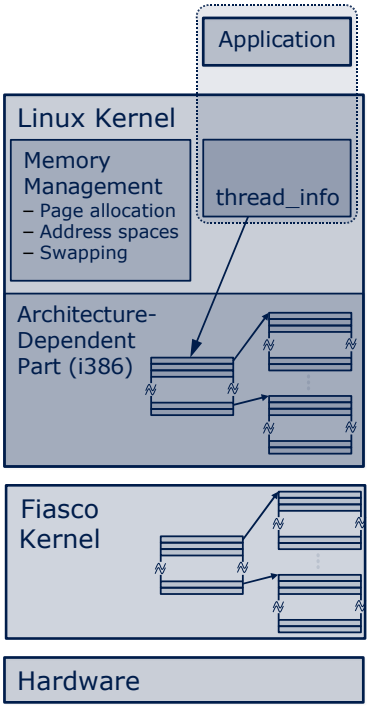
Linux address space management

- Architecture-independent part:
  - General page table management
  - Implements allocator strategies
  - Page replacement strategies
  - Assumes 4-level page table by architecture-dependent part
- Architecture-dependent part
  - Set, remove and test entries
  - TLB handling
  - Linux for x86 uses 2 level page tables

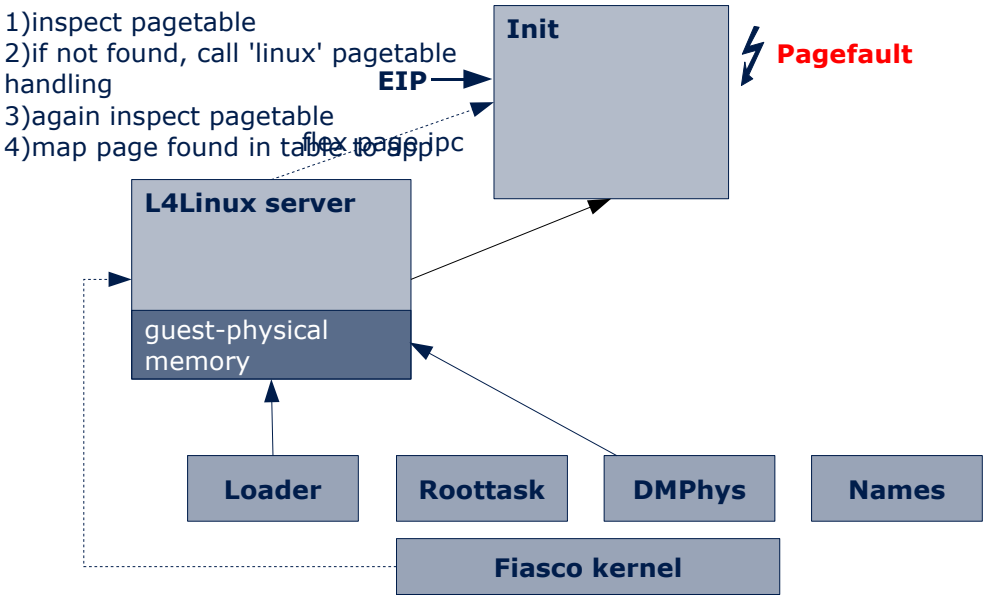


L4Linux address space management

- L4Linux user processes are actually L4 tasks
- L4Linux server is the pager
- Hardware page tables are managed by L4 kernel
- L4Linux page tables are mirrored from L4 for internal usage
  - L4Linux uses map/unmap operations
  - Adding page table entries is done lazy (pagefault occurs)



L4Linux address space management

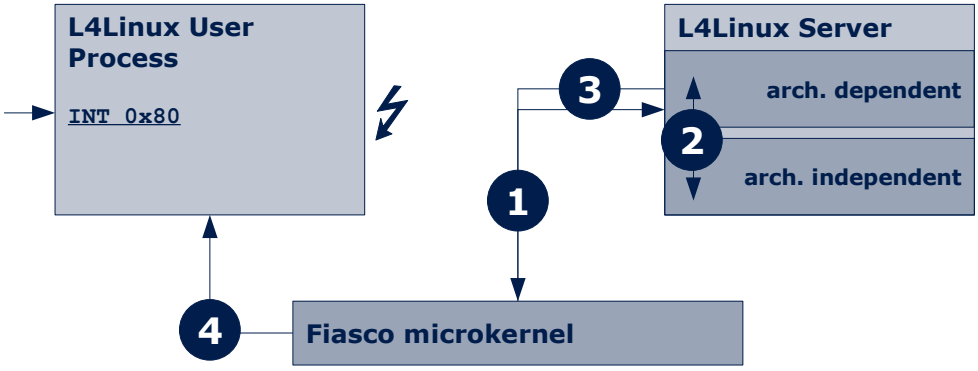


General exception handling

- If a L4 task raises an exception kernel sends exception IPC to handler (feature in Fiasco and L4.X2)
- Exception IPC contains CPU state of the client
- Exception handler can reply with a new state, for instance another instruction pointer
- Exception IPC can be used to recognize Linux system calls:
  - INT 0x80 will trigger an exception, due to lack of IDT gate in L4 kernel
  - L4Linux server acts as exception handler for its user processes

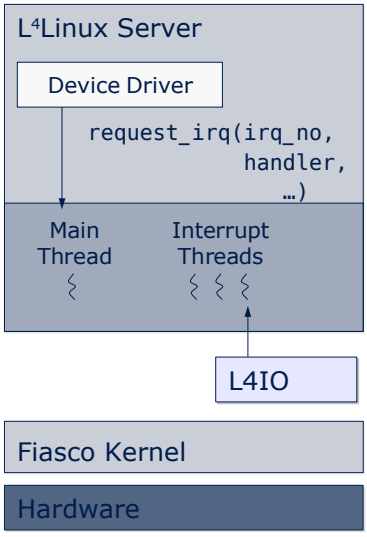
L4Linux kernel entry

- System call costs:
  - 2x kernel entry/exit (exception and reply)
  - 2x address space switch



Interrupt handling

- Interrupt messages are received in separate threads
- Interrupt threads run on a higher priority than other Linux threads (Linux semantic)
- Interrupt thread wake up idle thread or force the running user process to enter the linux server
- Plain Linux disables interrupts for syncronizaion
  - Use a lock instead of `CLI/STI`



Open issues

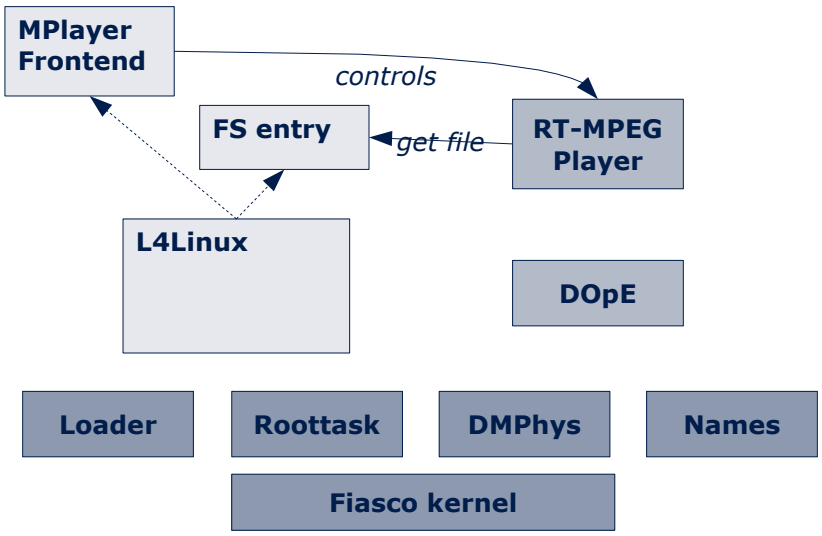
- Linux kernel needs to access address space of user processes (e.g. syscall arguments)
  - walk page tables of user process
- Security problems with DMA
  - move device drivers out of L4Linux
  - I/O MMU
- L4Linux has to schedule its tasks itself
  - only one L4Linux process is active at a time
  - other processes are waiting in IPC (exception or pagefault)

Hybrid applications

- Linux applications that are 'L4 aware'
- Needs to be detected by Linux server
  - Linux server puts them in UNINTERRUPTIBLE state in its own data structures
  - Will not disturb ongoing IPC in hybrid task
- L4Linux user processes run as *Aliens*
  - Special alien flag used when creating a task
  - Aliens trap when calling L4 system
  - Exception handler monitors system call
  - Fiasco-only feature

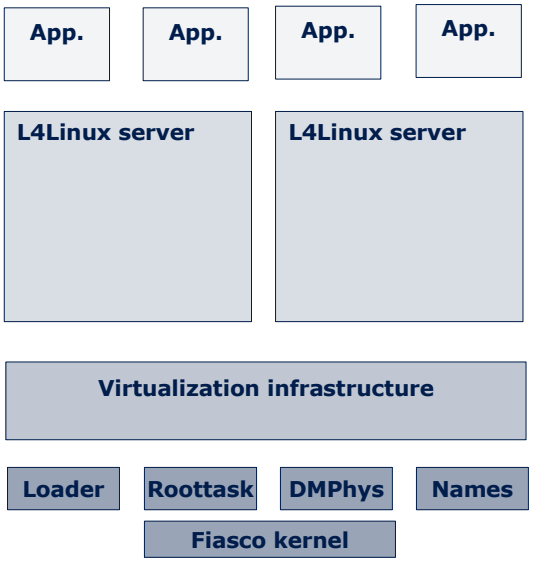
Real-time video player

- L4Linux user processes might use native L4 tasks or provide services themself



Multiple L4Linux instances

- Using multiple instances concurrently, e.g. for each security domain
- Devices need to be multiplexed (see resource management lesson: ORe, nitpicker)
- Communication through network, special IPC monitors ...

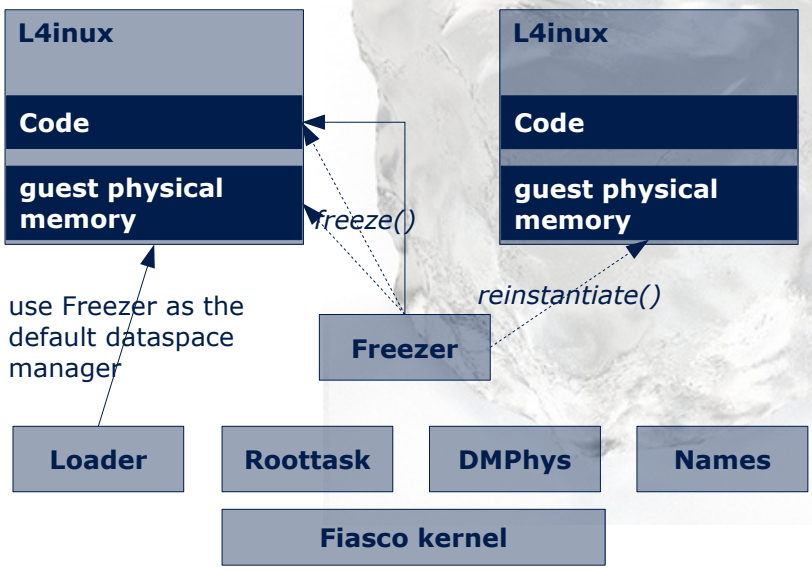


Freezing L4Linux

- Problem with several L4Linux instances:
  - Wasting memory
  - Long boot process
- Solution:
  - Freeze one L4Linux instance
  - Make copies when necessary
  - Use Dataspace manager interface to minimize necessary changes

Freezing L4Linux

- Pages are mapped read-only to new instance
  - Use copy on write (CoW)



Summary

- Virtualization flavors
  - API or ABI
  - Full or partial virtualization
  - Hardware (especially x86) or OS
- NOVA
  - Minimize hypervisor by taking out 'virtualization policy'
- L4Linux – paravirtualization in detail
  - Address space layout & management
  - Taming Linux (interrupts, I/O memory)
  - Freeze it

References

- Adam Lackorzynski: **'L4Linux Porting Optimizations'** Diploma Thesis 2004
- Keith Adams and Ole Agesen: **'A Comparision of Software and Hardware Techniques for x86 Virtualization'** ASPLOS 2006
- Udo Steinberg: **'NOVA Hypervisor Architecture Whitepaper'** Internal Report 2007
- **Intel Virtualization Technology**
  http://www.intel.com/technology/itj/2006/v10i3/1-hardware/1-abstract.htm
- **L4Linux Webpage**
  http://os.inf.tu-dresden.de/L4/LinuxOnL4

Outlook

- Tomorrow paper reading:
  - *Singularity – rethinking the software stack* a Microsoft Research project
  - as usual: read, understand and summarize it
- Next week virtualization part II:
  - Legacy containers through emulation
  - Libc or Qt on top of L4