



VIRTUALIZATION

Julian Stecklina (jsteckli@os.inf.tu-dresden.de)

Dresden, 2010/11/30

00 Goals

Give you an overview about:

- virtualization and virtual machines *in general*,
- hardware virtualization on x86,
- our research regarding virtualization.

We will *not* discuss:

- lots and lots of details,
- language runtimes,
- how to use XEN/KVM/. . .



00 Outline

What's Virtualization?

Very Short History

Virtualization on x86

Example: L4Linux

Example: NOVA



01 Outline

What's Virtualization?

Very Short History

Virtualization on x86

Example: L4Linux

Example: NOVA

01 The Hype



vmware®



Windows Server® 2008
Hyper-V™

01 Virtualization

virtual existing in essence or effect though not in actual fact

<http://wordnetweb.princeton.edu>

"All problems in computer science can be solved by another level of indirection."

Butler Lampson, 1972

01 Emulation

Suppose you develop for a system G (*guest*, e.g. an ARM-base phone) on your workstation H (*host*, e.g., an x86 PC). An *emulator* for V running on H precisely emulates G 's

- CPU,
- memory subsystem, and
- I/O devices.

Ideally, programs running on the emulated G exhibit the same behaviour as when running on a real G (except for timing).

01 Emulation (cont'd)

The emulator

- simulates every instruction in software as its executed,
- prevents direct access to H 's resources from code running inside G ,
- maps G 's devices onto H 's devices,
- may run multiple times on H .

01 Mapping G to H

Both systems may have considerably different

- instructions sets and
- hardware devices

making emulation slow and complex (depending on emulation fidelity).

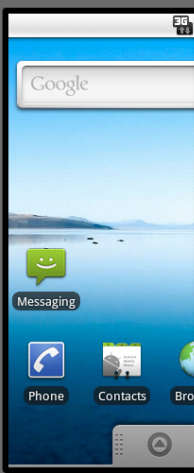
Android SDK and AVD Manager

- Virtual Devices
- Installed Packages
- Available Packages
- Settings
- About

List of existing Android Virtual Devices located at /home/julian/.android/avd

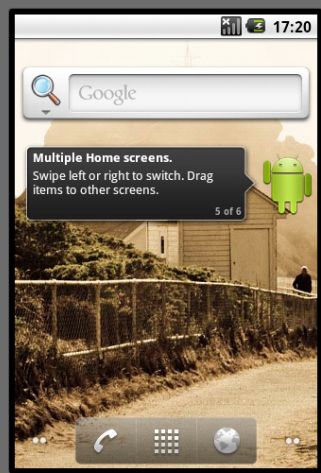
AVD	
5556:android22	
✓ an	
✓ an	

5554:android21




Home screen of an Android 2.1 emulator. It features a blue background with a white Google search bar at the top. Below the search bar is a green 'Messaging' app icon. At the bottom, there are icons for 'Phone', 'Contacts', and 'Browser'. The status bar at the top shows the time as 17:20.


5556:android22



Home screen of an Android 2.2 emulator. It features a landscape background with a wooden fence and a house. A green Android robot is on the right. A black text box in the center reads: "Multiple Home screens. Swipe left or right to switch. Drag items to other screens. 5 of 6". At the bottom, there are icons for 'Phone', 'App Drawer', and 'Browser'. The status bar at the top shows the time as 17:20.



Navigation controls for the Android emulator. It includes a camera icon, a speaker icon, a volume icon, a power icon, a green phone icon, a directional pad, a red phone icon, a home icon, a menu icon, a back icon, and a search icon. Below these is a full QWERTY keyboard with function keys like ALT, SYM, and @.








Another view of the Android emulator keyboard, showing the same layout as the one above, including the QWERTY keys and function keys.

```

43200k free, 12800k buffers
43548k free, 3711812k cached

CPU MEM TIME+ COMMAND
11 2.4 2:35.66 emulator
9 2.4 3:33.22 emulator
    
```

 Android SDK and AVD ...  5554:android21  5556:android22  julian@janus: ~ 

01 $G = H$

If host and virtualized hardware architecture is (about) the same,

- interpreting every executed instruction seems *not necessary*,
- near-native execution speed should be possible.

This is (easily) possible, if the architecture is *virtualizable*.

01 From Emulation to Virtualization

A **virtual machine** is defined to be an

“efficient, isolated duplicate of a real machine.”

(Popek, Goldberg 1974)

The software that provides this illusion is the *Virtual Machine Monitor* (VMM, mostly used synonymous with *Hypervisor*).

01 Virtualizable?

... is a property of the *Instruction Set Architecture* (ISA). Instructions are divided into two classes:

A *sensitive* instruction

- changes the configuration or mode of the processor, or
- depends in its behavior on the processor's state.

A *privileged* instruction

- causes a trap (unconditional control transfer to privileged mode) in user mode.

01 Trap & Emulate

If all sensitive instructions are privileged,
a VMM can be written.

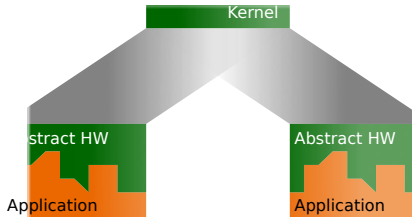
- execute guest in unprivileged mode,
- emulate all instructions that cause traps.

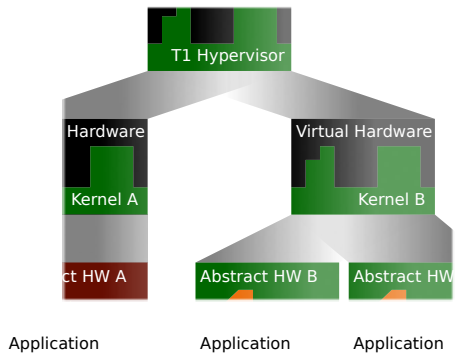
01 Trap & Emulate (cont'd)

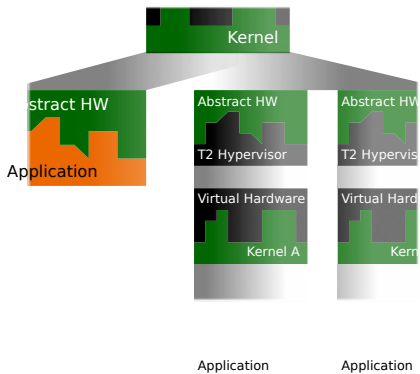
Will be topic of seminar on December 14th:

Formal Requirements for
Virtualizable Third-Generation Architectures
<http://portal.acm.org/citation.cfm?id=361073>

01 Where to put the VMM?







01 Type 1 vs. Type 2

Type 1 are implemented on the bare metal:

- no OS overhead
- complete control over host resources
- high maintainance effort

Popular examples are

- Xen,
- VMware ESXi.

01 Type 1 vs. Type 2 (cont'd)

Type 2 run as normal process on top of an OS:

- doesn't reinvent the wheel
- performance may suffer

Popular examples are

- KVM,
- VMware Server/Workstation,
- VirtualBox,
- ...

01 Paravirtualization

Why all the trouble? Just “port” a guest operating system to the interface of your choice.

Paravirtualization can

- provide better performance,
- simplify VMM

but at a maintainance cost and you need the source code!

Compromise: Use paravirtualized drivers for I/O performance (KVM virtio, VMware).

Examples are MkLinux, L4Linux, Xen, ...

01 Reimplementation of the OS Interface

E.g. *wine* reimplements (virtualizes) the Windows ABI.

- Run unmodified Windows binaries.
- Windows API calls are mapped to Linux/FreeBSD/Solaris/MacOS X equivalents.
- Huge moving target!

Can also be used to recompile Windows applications as native applications linking to *winelib* \Rightarrow API “virtualization”

01 Recap

- *Virtualization* is an overloaded term. Classification criteria:
 - **Target**
real hardware, OS API, OS ABI, ...
 - **Emulation vs. Virtualization**
Interpret some or all instructions?
 - **Guest Modifications?**
Paravirtualization

01 Recap (cont'd)

- A (Popek/Goldberg) *Virtual Machine* is an
 - efficient,
 - isolated
 - duplicate of a real machine.
- The software that implements the VM is the *Virtual Machine Monitor* (hypervisor).
- Type 1 hypervisors run on the bare metal.
- Type 2 hypervisors run as applications on a conventional OS.

02 Outline

What's Virtualization?

Very Short History

Virtualization on x86

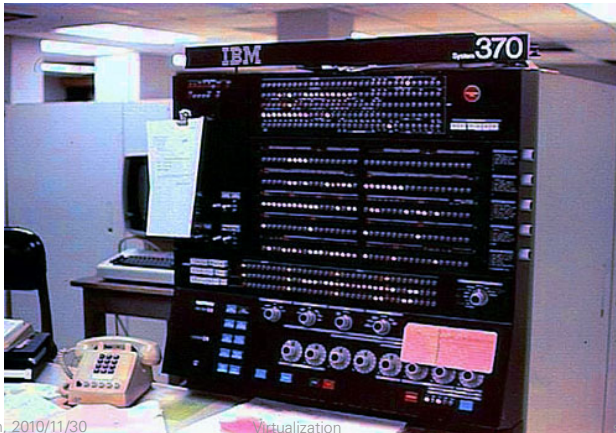
Example: L4Linux

Example: NOVA

“Virtual machines have finally arrived. Dismissed for a number of years as merely academic curiosities, they are now seen as cost-effective techniques for organizing computer systems resources to provide extraordinary system flexibility and support for certain unique applications.”

Survey of Virtual Machine Research
Robert P. Goldberg
1974

02 Early History: IBM



02 Early History: IBM

Virtualization was pioneered with IBM's CP/CMS in ~1967 running on System/360 and System/370:

- CP Control Program
provided System/360 virtual machines.

- CMS Cambridge Monitor System (later Conversational Monitor System)
single-user OS.

At the time more flexible and efficient than time-sharing multi-user systems.

02 Early History: IBM (cont'd)

CP encodes guest state in a hardware-defined format.

SIE Start Interpretive Execution (instruction)
runs the VM until a trap or interrupt occurs. CP resume control
and handles trap.

CP provides:

- memory protection between VMs,
- preemptive scheduling.

Gave rise to IBM's VM line of operating systems.

First release: 1972

Latest release: z/VM 6.1 (2009)

02 Virtualization is Great

- Consolidation
 - improve server utilization
- Isolation
 - isolate services for security reasons or
 - because of incompatibility
- Reuse
 - run legacy software
- Development

...but is confined to the mainframe world for a very long time.

... fast forward to the late nineties ...



03 Outline

What's Virtualization?

Very Short History

Virtualization on x86

Example: L4Linux

Example: NOVA

03 Is x86 Virtualizable?

x86 has several virtualization holes that violate Popek&Goldberg requirement.

- Possibly too expensive to trap on every privileged instruction.
- `popf` (pop flags) silently ignores writes to the Interrupt Enable flag in user mode. Should trap!
- More in the seminar.

03 VMware Workstation: Binary Translation

First commercial virtualization solution for x86, introduced in ~1999. Overcame limitations of the x86 architecture:

- translate problematic instructions into appropriate calls to the VMM
- can avoid costly traps for privileged instructions

Provided decent performance but:

- requires complex runtime translation engine (self-modifying code...)

Other examples: KQemu, Virtual Box, Valgrind

03 Hardware Support for Virtualization

Late Pentium 4 (2004) introduced hardware support for virtualization: Intel VT.
(AMD-V is conceptually very similar)

- root mode vs. non-root mode
 - root mode runs hypervisor
 - non-root mode runs guest
- situations that Intel VT cannot handle trap to root mode (**VM Exit**)
- special memory region (VMCS) holds guest state
- reduced software complexity

Supported by all major virtualization solutions today.

03 Instruction Emulator

Intel VT and AMD-V still require an instruction emulator, e.g. for

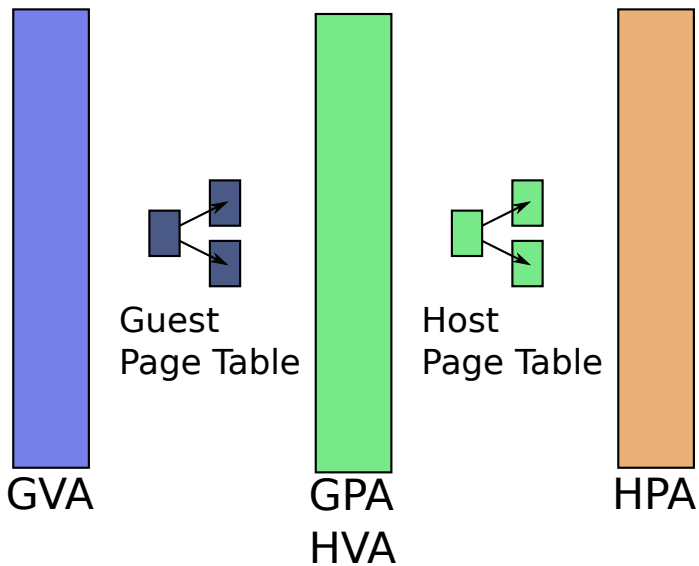
- running 16-bit code (not in AMD-V, latest Intel VT),
 - BIOS
 - boot loaders
- handling MMIO (need to emulate instruction that caused a page fault)
 - realized as non-present page in guest physical mode
 - emulate offending instruction
- ...

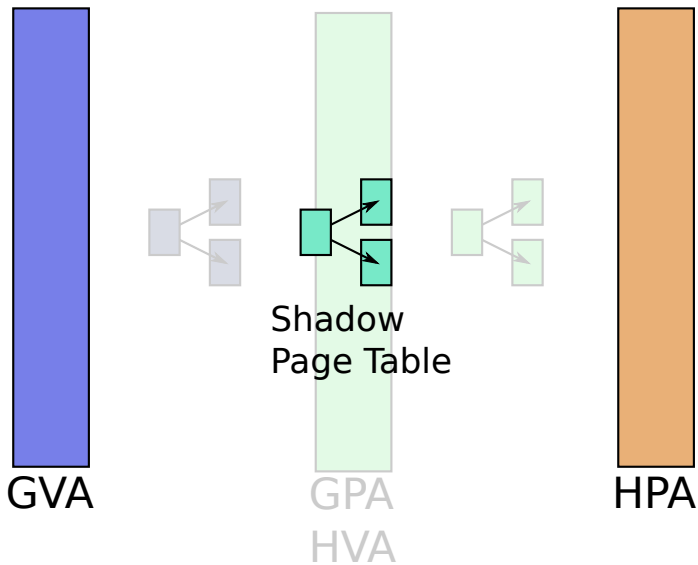
03 MMU Virtualization

Early version of Intel VT do not completely virtualize the MMU. VMM has to handle paging in guest.

Four different types of memory addresses:

- HPA Host Physical Address
- HVA Host Virtual Address
- GPA Guest Physical Address
- GVA Guest Virtual Address





03 Drawbacks of Shadow Paging

Maintaining Shadow Page Tables causes significant overhead, because they need to be updated/recreated on

- guest page table modification,
- guest address space switch.

03 Shadow Paging

1. page fault in guest (GVA)
2. traps to VMM
3. parse guest page tables (GVA \Rightarrow GPA)
4. maybe inject page fault to guest (no mapping found, true pagefault)
5. translate to shadow page table entry (GPA \Rightarrow HVA \Rightarrow HPA)
6. create entry in shadow page table
7. resume guest

03 Nested Paging

Introduced in the Intel *Nehalem* (EPT) and AMD *Barcelona* (Nested Paging) microarchitectures, the CPU can handle

- guest and
- host page table

at the same time. Can reduce VM Exits by *two orders of magnitude*, but introduces

- measurable constant overhead ($< 1\%$)

03 Nested Paging (cont'd)

Event	Nested Paging	Shadow Paging
vTLB Fill		181,966,391
Guest Page Fault		13,987,802
CR Read/Write		3,000,321
vTLB Flush		2,328,044
INVLPG		537,270
Hardware Interrupts	174,558	239,142
Port I/O	610,589	723,274
Memory-Mapped I/O	76,285	75,151
HLT	3,738	4,027
Interrupt Window	2,171	3,371
Sum	867,341	202,864,793
Runtime (seconds)	470	645

(Linux Kernel Compile)



04 Outline

What's Virtualization?

Very Short History

Virtualization on x86

Example: L4Linux

Example: NOVA

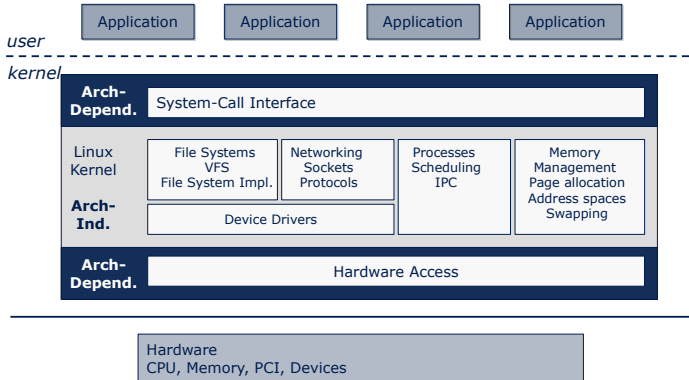
04 L4Linux

... is a paravirtualized Linux first presented at SOSP'97 running on the original L4 kernel.

- L4Linux predates the x86 virtualization hype
- L4Linux 2.2 supported MIPS and x86
- L4Linux 2.4 first version to run on L4Env
- L4Linux 2.6 uses Fiasco.OC's paravirtualization features

The current status:

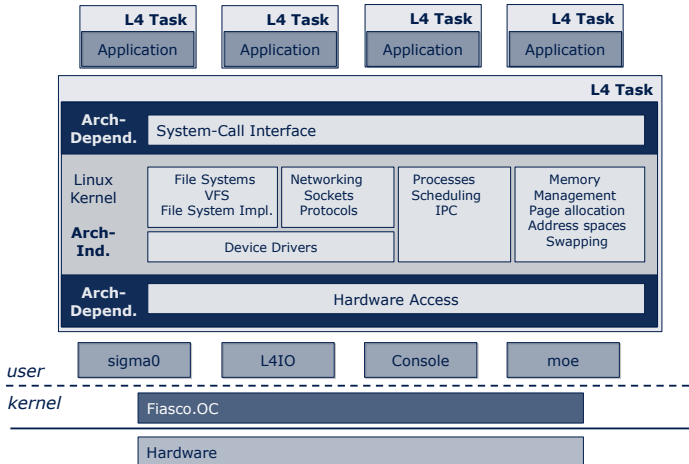
- Linux 2.6.36
- x86 and ARM support
- SMP



04 Porting Linux to L4

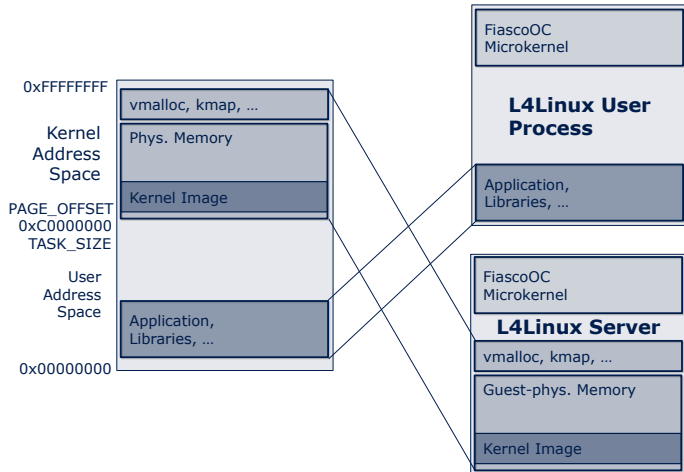
Regard L4 as new hardware platform. Port small architecture dependent part:

- system call interface
 - kernel entry
 - signal delivery
 - copy from/to user space
- hardware access
 - CPU state and features
 - MMU
 - interrupts
 - memory-mapped and port I/O



04 L4Linux Architecture

- L4 specific code is divided into:
 - x86 and ARM specific code
 - hardware generic code
- Linux kernel and Linux user processes run each with a single L4 task.
 - L4Linux kernel task does not see a L4Linux process virtual memory



04 L4Linux Challenges

The L4Linux kernel “server” has to:

- access user process data,
- manage page tables of its processes,
- handle exceptions from processes, and
- schedule them.

L4Linux user processes have to:

- “enter” the L4Linux kernel (living in a different address space).

04 Kernel Entry

Normal Linux syscall interface (int 80h) causes trap.

- L4Linux server receives exception IPC.

Heavyweight compared to native Linux system calls:

- two address space switches,
- two Fiasco kernel entries/exits

04 Interrupts

L4Linux has a thread for each virtual interrupt.

- Interrupts are received as messages.
- Interrupt threads have higher priority than normal Linux threads (Linux semantics).
- Interrupt threads force running user process (or idle thread) into L4Linux server.
- Linux uses CLI/STI to disable interrupts, L4Linux uses a lock.

04 vCPUs

Simplify interrupt/exception handling by introducing vCPUs (Fiasco.OC):

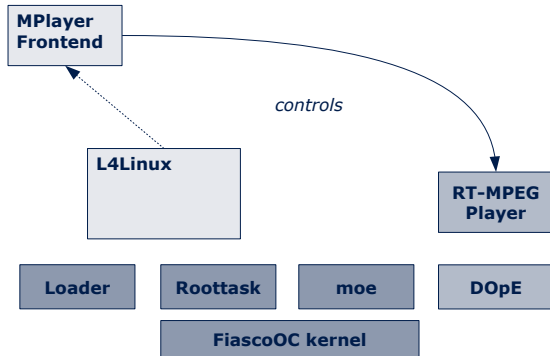
- have dedicated interrupt entry points,
 - need to differentiate between interrupt and systemcall
- can be rebound to different tasks,
 - simulates address space switches
- can mask interrupts
 - emulates Interrupt Enable flag
 - don't need that lock anymore

04 Not Covered in Detail

- access to user process' memory
 - walk page tables of user process
- device drivers
 - DMA is problematic
 - IOMMU
- scheduling
 - only one user process active at any time

04 Hybrid Applications

- Linux applications that are “L4 aware”
- Need special handling by L4Linux server
 - processes with ongoing IPC are marked UNINTERRUPTIBLE in L4Linux
 - IPC is not disturbed



04 Multiple L4Linux Instances

Of course, you can run multiple L4Linux instances.

- isolate applications for e.g. security reasons,
- communication via network or IPC
- devices need to be multiplexed (will be covered in resource management lecture)

04 L4Linux as Toolbox

Reuse large parts of code from Linux:

- filesystems,
- network stack,
- device drivers.

Use a hybrid application to provide this service to “native” L4 applications.



05 Outline

What's Virtualization?

Very Short History

Virtualization on x86

Example: L4Linux

Example: NOVA

05 Security and Virtualization

Virtualizing several insecure operating systems on one host is *safe* and *secure*, because

- they cannot interfere with each other, and
- they are confined to their virtual machine.

Is it? Does virtualization add security?

05 Recap

Xen runs its instruction emulator and virtual devices in Ring 0 in Root Mode.

KVM runs its instruction emulator and virtual devices (mostly) as normal Linux application in Ring 3 in Root Mode.

...but both share code for instruction emulation and virtual devices.

05 Secunia Advisory SA25073

<http://secunia.com/advisories/25073/>

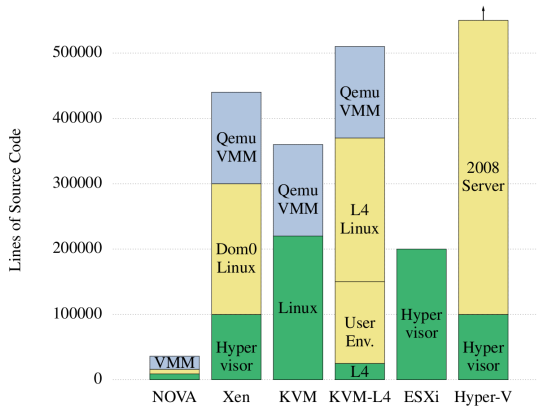
- “The size of ethernet frames is not correctly checked against the MTU before being copied into the registers of the NE2000 network driver. This can be exploited to cause a heap-based buffer overflow.”
- “ An error within the handling of the aam instruction can result in a division by zero.”
- ...

05 TCB of Virtual Machines

The *Trusted Computing Base* of a Virtual Machine is the amount of hardware and software you have to trust to guarantee this VM's security. (More in lecture on Security)

For e.g. KVM this (conservatively) includes:

- the Linux kernel,
- Qemu.

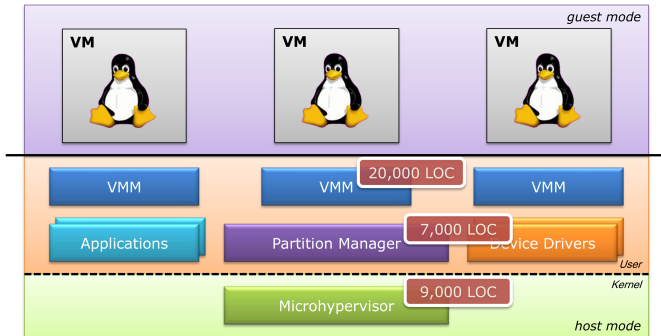


05 NOVA Architecture

Reduce complexity of hypervisor:

- hypervisor provides low-level protection domains
 - address spaces
 - virtual machines
- VM exits are relayed to VMM as IPC with selective guest state,
- one VMM per guest in (root mode) userspace,
 - possibly specialized VMMs to reduce attack surface
 - only one generic VMM implement so far

05 NOVA OS Virtualization Architecture



05 Example: Low Complexity VMM

Diplomarbeit by Steffen Liebergeld

Idea: Reduce TCB of VMM by using paravirtualization *and* hardware-assisted virtualization.

- Implemented on Fiasco using AMD-V
- Small VMM: 3800 LOC
- 300 LOC changed in Linux
- No instruction emulator required
 - no MMIO
 - no 16-bit code
- Only simple paravirtualized device models required: 2600 LOC
 - salvaged from L4Linux

05 Recap: Examples

- L4Linux is the paravirtualized workhorse on L4:
 - reuse Linux applications,
 - reuse Linux components.
- NOVA provides faithful virtualization with small TCB for VMs.

05 Next Weeks

Today's seminar (IPC) is in

INF E042

Next week's lecture will be about Resource Management.

Don't forget to read until **December 14th**:

Formal Requirements for
Virtualizable Third-Generation Architectures
<http://portal.acm.org/citation.cfm?id=361073>

05 References

<https://www.cs.ucsb.edu/~ravenben/papers/coreos/Gol74.pdf>
<http://www.diku.dk/hjemmesider/ansatte/jacobg/thesis.pdf>
http://os.inf.tu-dresden.de/papers_ps/steinberg_eurosys2010.pdf
<http://os.inf.tu-dresden.de/pubs/sosp97/>