

Open Kernel Labs™

The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors

Gernot Heiser, Ben Leslie

Open Kernel Labs and NICTA and UNSW
Sydney, Australia



Microkernels vs Hypervisors

- > Hypervisors = “microkernels done right?” [Hand et al, HotOS '05]
 - Talks about “liability inversion”, “IPC irrelevance” ...
- > What’s the difference anyway?

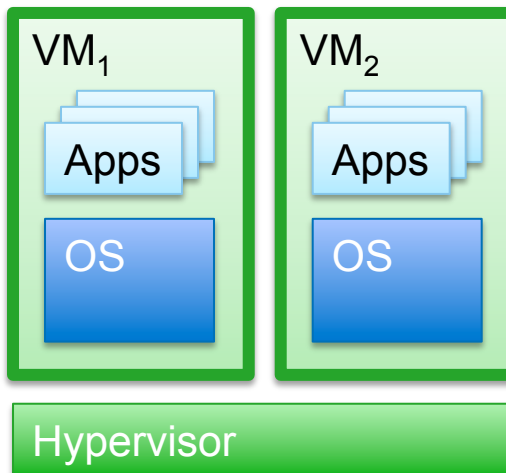


ok-labs.com

©2010 Open Kernel Labs and NICTA. All rights reserved.

What are Hypervisors?

- > Hypervisor = “virtual machine monitor”
 - Designed to multiplex multiple virtual machines on single physical machine

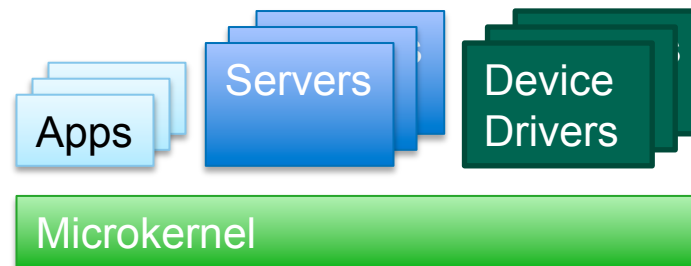


- > Invented in '60s to time-share with single-user OSes
- > Re-discovered in '00s to work around broken OS resource management



What are Microkernels?

- > Designed to minimise kernel code
 - Remove policy, services, retain mechanisms
 - Run OS services in user-mode
 - Software-engineering and dependability reasons
 - L4: ≈ 10 kLOC, Xen ≈ 100 kLOC, Linux: $\approx 10,000$ kLOC



- > IPC performance critical (highly optimised)
 - Achieved by API simplicity, cache-friendly implementation
- > Invented 1970 [Brinch Hansen], popularised late '80s (Mach, Chorus)



ok-labs.com

©2010 Open Kernel Labs and NICTA. All rights reserved.

What's the Difference?

- > Both contain all code executing at highest privilege level
 - Although hypervisor may contain user-mode code as well
- > Both need to abstract hardware resources
 - Hypervisor: abstraction closely models hardware
 - Microkernel: abstraction designed to support wide range of systems
- > What must be abstracted?
 - Memory
 - CPU
 - I/O
 - Communication



What's the difference?

Resource	Hypervisor	Microkernel
Memory	Virtual MMU (vMMU)	Address space
CPU	Virtual CPU (vCPU)	Thread or scheduler activation
I/O	<ul style="list-style-type: none"> • Simplified virtual device • Driver in hypervisor • Virtual IRQ (vIRQ) 	<ul style="list-style-type: none"> • IPC interface to user-mode driver • Interrupt IPC
Communication	Virtual NIC, with driver and network stack	High-performance message-passing IPC

Just page tables in disguise

Just kernel-scheduled activities

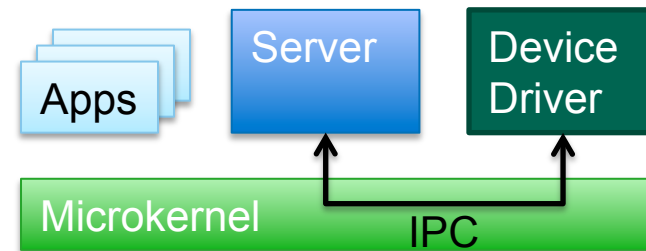
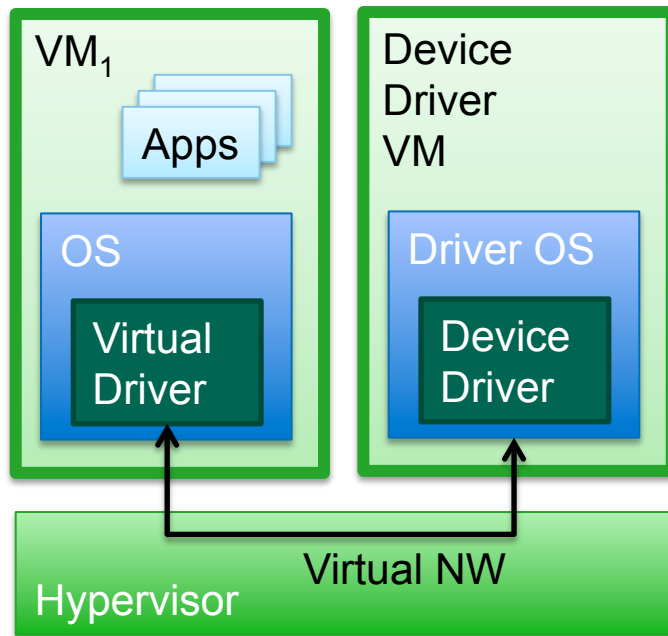
Real Difference?

Modelled on HW, Re-uses SW

Minimal overhead, Custom API



Closer Look at I/O and Communication



- > Communication is critical for I/O
 - Microkernel IPC is highly optimised
 - Hypervisor inter-VM communication is frequently a bottleneck



Hypervisors vs Microkernels: Summary

- > Fundamentally, both provide similar abstractions
- > Optimised for different use cases
 - Hypervisor designed for virtual machines
 - API is hardware-like to ease guest ports
 - Microkernel designed for multi-server systems
 - seems to provide more OS-like abstractions



ok-labs.com

©2010 Open Kernel Labs and NICTA. All rights reserved.

Hypervisors vs Microkernels: Drawbacks

Hypervisors:

- > Communication is Achilles heel
 - More important than expected
 - Critical for I/O
 - Plenty attempts to improve in Xen
- > Most hypervisors have big TCBs
 - Infeasible to achieve high assurance of security/safety
 - In contrast, microkernel implementations can be *proved* correct

Microkernels:

- > Not ideal for virtualization
 - API not very effective
 - L4 virtualization performance close to hypervisor
 - effort much higher
 - Virtualization needed for legacy
- > L4 model uses kernel-scheduled threads for more than exploiting parallelism
 - Kernel imposes policy
 - Alternatives exist, eg. K42 uses scheduler activations

Could we get the best of both?

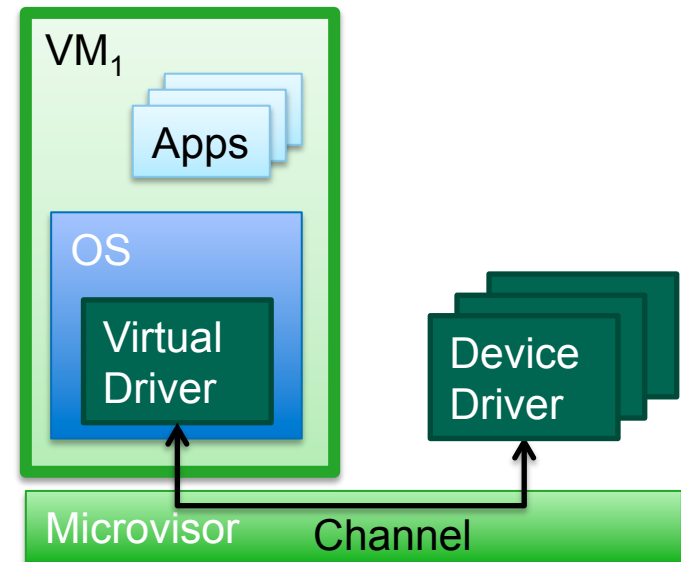


ok-labs.com

©2010 Open Kernel Labs and NICTA. All rights reserved.

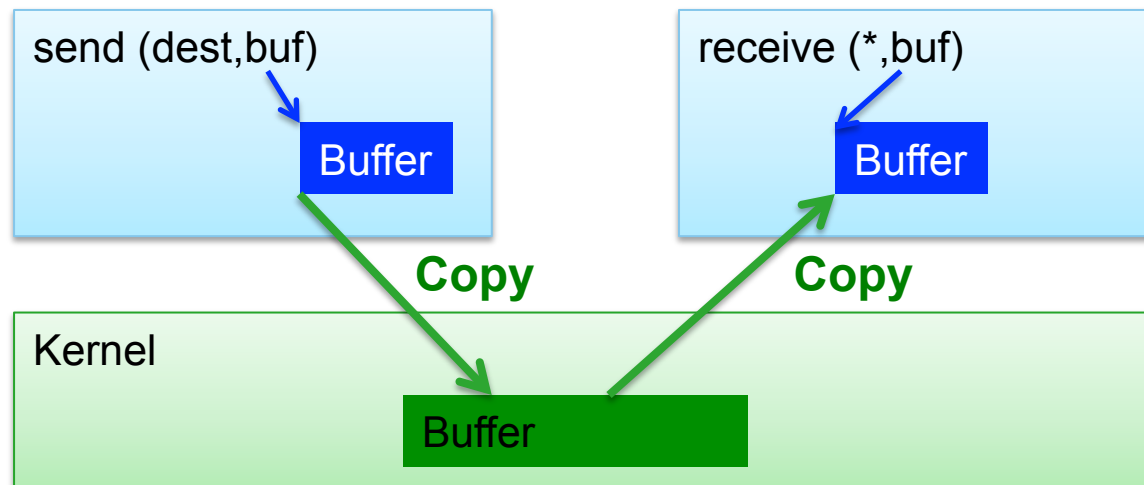
Best of Both Worlds: The OKL4 Microvisor

- > Microvisor = microkernel hypervisor
- > Microkernel with a hypervisor API:
 - **vCPU**: scheduled by microvisor on real CPU
 - Multiple vCPUs per VM for running on multiple cores
 - **vMMU**: looks like a large TLB
 - caches mappings
 - **vIRQ**: virtualize interrupts and provide asynchronous inter-VM signals
- > Plus asynchronous **channel** abstraction
 - for high-bandwidth, low-latency communication
- > Small TCB: ≈ 10 kLOC
- > Suitable for multi-server systems



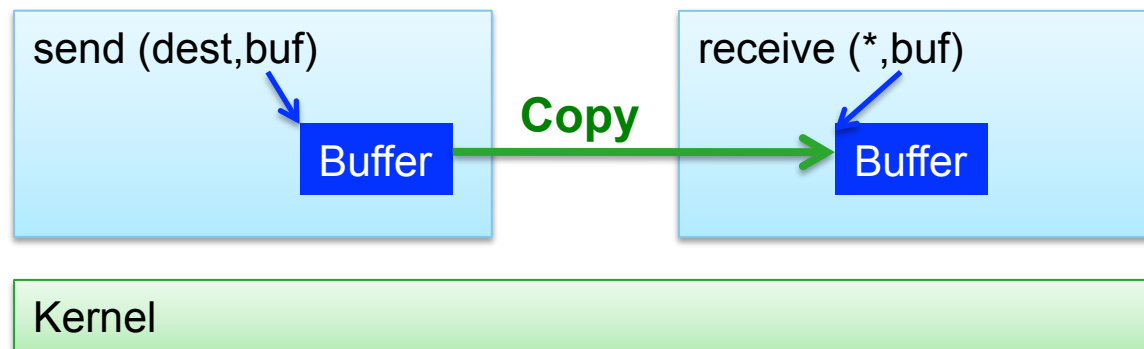
But Liedtke Said: “IPC Shall Be Synchronous!”

- > Early microkernels had poorly-performing semi-synchronous IPC
 - Asynchronous send, synchronous receive
 - Requires message buffering in kernel
 - Each message is copied twice!



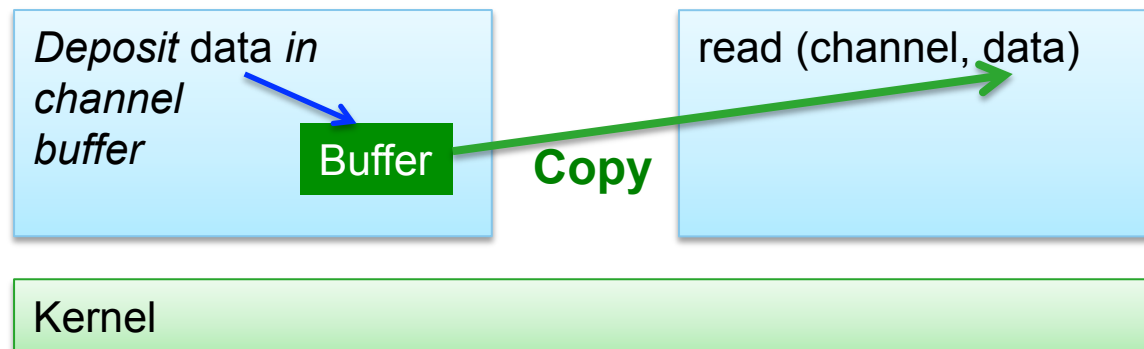
But Liedtke Said: “IPC Shall Be Synchronous!”

- > Early microkernels had poorly-performing semi-synchronous IPC
- > Liedtke designed L4 IPC fully synchronously to avoid such overheads
 - Rendezvous model: each operation blocks until partner is ready
 - No buffering, message copied directly from sender to receiver
 - Requires kernel-scheduled threads to avoid blocking whole app!



OKL4 Microvisor Communication is Asynchronous

- > In our experience, real-world embedded systems require asynchronous IPC
 - Maps more naturally to hardware events
 - In the past, we added asynchronous notification (lightweight signals) to L4 IPC
- > The OKL4 Microvisor only provides fully asynchronous communication:
 - vIRQs as asynchronous signals
 - Channels for asynchronous data transfer
 - Still single-copy, no kernel buffers!
 - Synchronised eg. by sending vIRQ



Performance: Imbench

Benchmark	Native	Virtualized	Overhead		
null syscall	0.6 μ s	0.96 μ s	0.36 μ s	180 cy	60 %
read	1.14 μ s	1.31 μ s	0.17 μ s	85 cy	15 %
stat	4.73 μ s	5.05 μ s	0.32 μ s	160 cy	7 %
fstat	1.58 μ s	2.24 μ s	0.66 μ s	330 cy	42 %
open/close	9.12 μ s	8.23 μ s	-0.89 μ s	-445 cy	-10 %
select(10)	2.62 μ s	2.98 μ s	0.36 μ s	180 cy	14 %
sig handler	1.77 μ s	2.05 μ s	0.28 μ s	140 cy	16 %
pipe latency	41.56 μ s	54.45 μ s	12.89 μ s	6.4 kcy	31 %
UNIX socket	52.76 μ s	80.90 μ s	28.14 μ s	14 kcy	53 %
fork	1,106 μ s	1,190 μ s	84 μ s	42 kcy	8 %
fork+execve	4,710 μ s	4,933 μ s	223 μ s	112 kcy	5 %
system	7,583 μ s	7,796 μ s	213 μ s	107 kcy	3 %



ok-labs.com

©2010 Open Kernel Labs and NICTA. All rights reserved.

Beagle Board (500 MHz ARMv7)

Performance: Netperf on Loopback Device

Type	Benchmark	Native	Virtualized	Overhead
TCP	Throughput	651 [Mib/s]	630 [Mib/s]	3 %
	CPU load	99 [%]	99	0 %
	Cost	12.5 [μ /KiB]	12.9 [μ s/KiB]	3 %
UDP	Throughput	537 [Mib/s]	516 [Mib/s]	4 %
	CPU load	99	99	0 %
	Cost	15.2	15.8 [μ s/KiB]	4 %

Beagle Board (500 MHz ARMv7)



ok-labs.com

©2010 Open Kernel Labs and NICTA. All rights reserved.

Conclusions

- > Classical hypervisors and microkernels both have significant drawbacks
 - Hypervisor inter-VM comms primitives are slow, hurts I/O
 - Also have huge TCBs
 - Microkernel APIs not ideal for virtualization
- > Microvisor combines best of both
 - Virtualization-oriented API
 - Microkernel (i.e. small) code base
 - Fast communication channels for multi-server designs (stand-alone drivers)
- > OKL microvisor shows excellent virtualization performance
- > Suitability for multi-server designs still needs to be proven

