NICTA

Do Microkernels Suck?

Gernot Heiser
UNSW, NICTA and Open Kernel Labs

Australian Government
Department of Communications, Information Technology and the Arts
Australian Research Council

NICTA Members
ANU THE AUSTRALIAN NATIONAL UNIVERSITY
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW First for Business Department of State and Regional Development

NICTA Partners
Victoria The Place To Be
THE UNIVERSITY OF MELBOURNE
The University of Sydney
Queensland Government
Griffith UNIVERSITY
QUT Queensland University of Technology
THE UNIVERSITY OF QUEENSLAND AUSTRALIA

OLS 2007

NICTA

- Talk by Christoph Lameter: “Extreme High Performance Computing or Why Microkernels Suck”
- Contents:
  - This is how we got Linux to scale to 1000's of CPUs
    - clearly knows what he's talking about
    - no need to add to this...
  - This is why microkernels can't do the same
    - clearly hasn't got a clue about microkernels
- I'll explain...

Summary of Paper

- Look, we've scaled Linux to 1000 processors [with a little help of billions of $$ from IBM, HP, SGI, ...], microkernels [developed mostly by cash-strapped universities] haven't done the same, obviously they suck
- Equivalent statement in 1998: Look, Windows has drivers for zillions of devices, Linux doesn't, hence Linux sux.
- Very scientific approach, right?
- OK, I'm exaggerating somewhat, but let's see what it really says...

Common Misconceptions

NICTA

- Microkernel-based systems are less reliable, as failure of one component makes whole system fail
- Wrong!
  - Counter example: QNX High Availability Toolkit (sold commercially since 2001)
  - More recent counter example: Minix 3, which is open source — check it out for yourself
- Were reliability matters most, microkernels are used
  - aerospace, automotive, medical devices...

A Voice from the Coal Face

- “NTFS-3G is a user/hybrid-space driver”
- “Similar functionality and performance on commodity hardware as in-kernel file systems”
- “The invested effort and resource were only a fraction of what is usually needed, besides other benefits.”
- “The empirical learnings keep being highly instructive, refuting widely believed folklore”

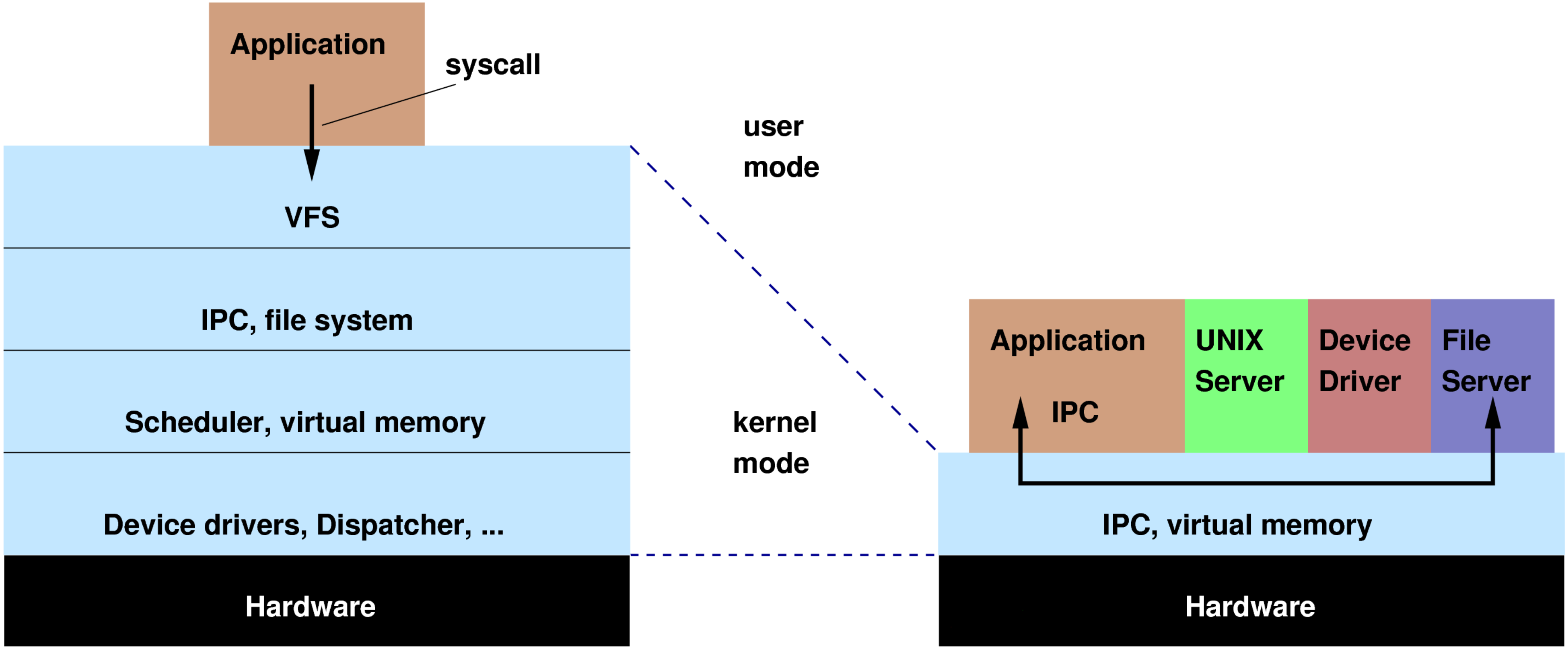
Szaka Szabolcs, leader of NTFS-3G, http://ntfs-3g.org

Common Misconceptions

- Microkernel relies on IPC, IPC requires expensive message queue operations, hence IPC is costly
- Wrong!
  - Counter example: L4, since 1993 (publ in SOSP)
  - L4 runs in 10s of millions of mobile phone
  - OS performance is critical for cell-phone baseband processing
  - L4 expected to run on 250M mobile devices within a year
- Why the sudden popularity?
  - it's fast
  - it's small
  - it enables fault containment

Let's Look at IPC

- IPC is used to obtain system service
  - IPC performance is important



Intrinsic Difference Syscall vs IPC

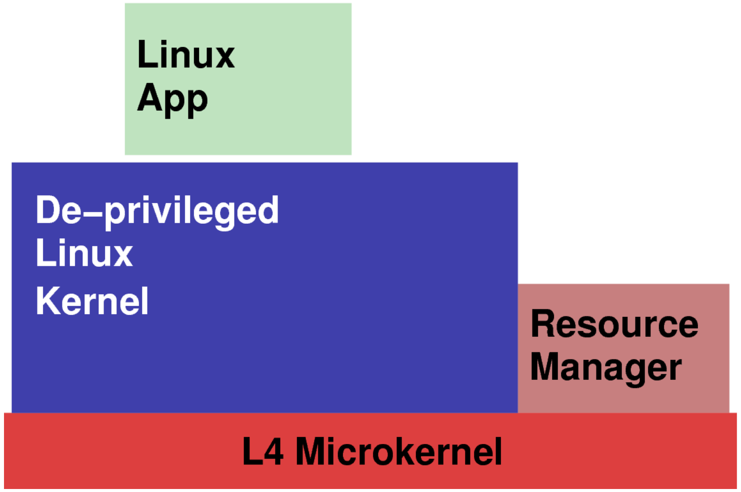
- Syscall: 2 mode switches (user→kernel, kernel→user)
- IPC: 2 mode switches + 1 context switch
- Server invocation needs 2 IPCs
  - extra cost is 2 mode switches, 2 context switches
- This is the inherent microkernel overhead!
  - it is wrong to think that IPC was used inside the system a lot (replacing function calls)
- Is it significant?
  - depends on the ratio between overhead and total cost of service obtained
  - it's a killer for the null system call
  - it's irrelevant for most others

Actual L4 IPC Cost [cycles]

Architecture	Intra address space	Inter address space
Pentium	113	305
AMD-64	125	230
Itanium	36	36
MIPS64	109	109
ARM Xscale	170	180

- How do a couple hundred cycles compare to the typical Linux system call???

Sort-of Extreme Example: Linux on L4



- Cops the full microkernel overhead
- Doesn't get any of the microkernel benefits
- How does it perform?

Linux on L4: ReAIM Macrobenchmark

NICTA

ReAIM Benchmark	Native	Virtualised	Ratio
1 Task	45.2	43.6	0.96
2 Tasks	23.6	22.6	0.96
3 Tasks	15.8	15.3	0.97

Native Linux vs Linux virtualized on L4

on Xscale PXA255 @ 400MHz

Not everything in L4 fully optimised yet (fork/exec)

Lmbench microbenchmarks

Benchmark	Native	Virtualized	Ratio
lmbench latencies in microseconds, smaller is better			
lat_proc procedure	0.21	0.21	0.99
lat_proc fork	5679	8222	0.69
lat_proc exec	17400	26000	0.67
lat_proc shell	45600	68800	0.66
lmbench bandwidths, MB/s, larger is better			
bw_file_rd 1024 io_only	38.8	26.5	0.68
bw_mmap_rd 1024 mmap_only	106.7	106	0.99
bw_mem 1024 rd	416	412.4	0.99
bw_mem 1024 wr	192.6	191.9	1
bw_mem 1024 rdwr	218	216.5	0.99
bw_pipe	7.55	20.64	2.73
bw_unix	17.5	11.6	0.66

Native Linux vs Linux virtualized on L4

on Xscale PXA255 @ 400MHz

Not everything in L4 fully optimised yet (fork/exec)

Lmbench Context Switching

Benchmark	Native	Virtualized	Ratio
lmbench latencies in microseconds, smaller is better			
lat_ctx -s 0 1	11	20	0.55
lat_ctx -s 0 2	262	5	52.4
lat_ctx -s 0 10	298	45	6.62
lat_ctx -s 4 1	48	58	0.83
lat_ctx -s 4 10	419	203	2.06
lat_fifo	509	49	10.39
lat_pipe	509	49	10.39
lat_unix	1015	77	13.18
lat_syscall null	0.8	4.8	0.17

Native Linux vs Linux virtualized on L4

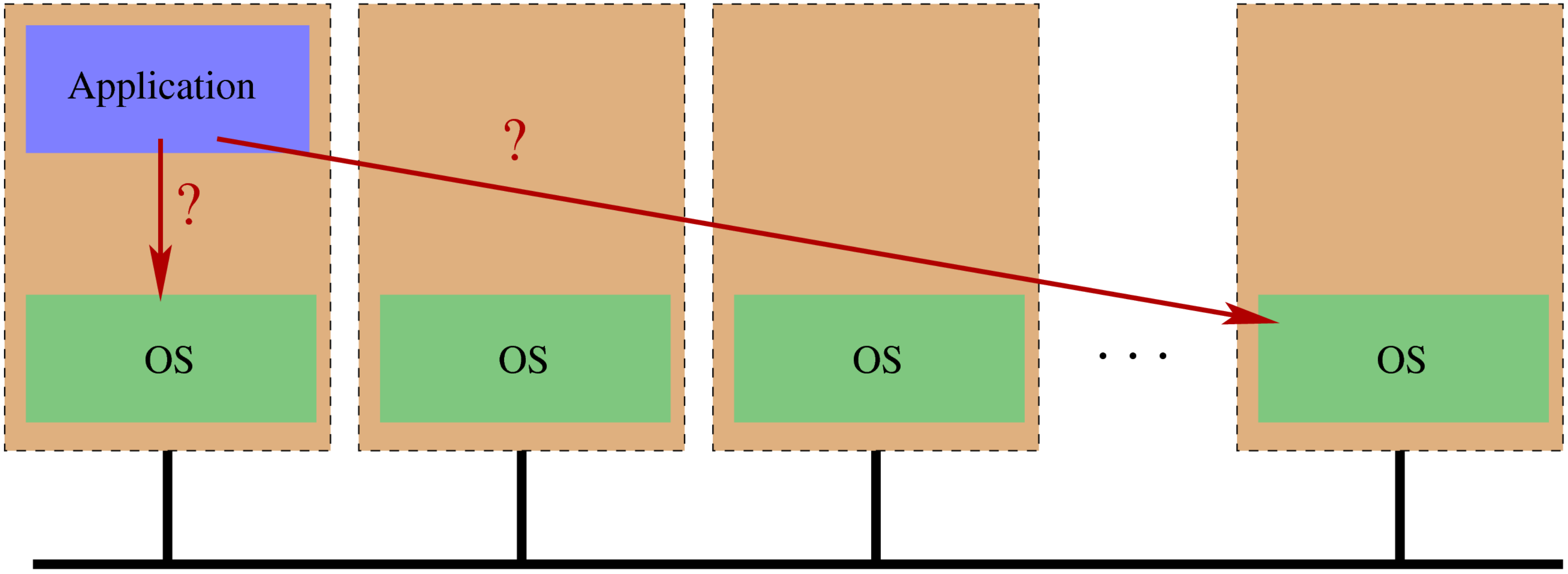
on Xscale PXA255 @ 400MHz

How Can Virtual be Faster than Real?

- It's a **microkernel**!
  - Complete kernel is about 10–11kloc!
- Linux is **big**!
  - 100s of kloc not counting drivers, file systems etc
- ARM MMU is quirky, needs a lot of effort to optimise
  - **much easier to optimize a small code base**
- Of course, the same can be achieved with Linux
  - in fact, we did it and offered patches upstream
  - maintainers didn't take — who cares about factor of 50!
  - Snapgear is running our patches in their modems

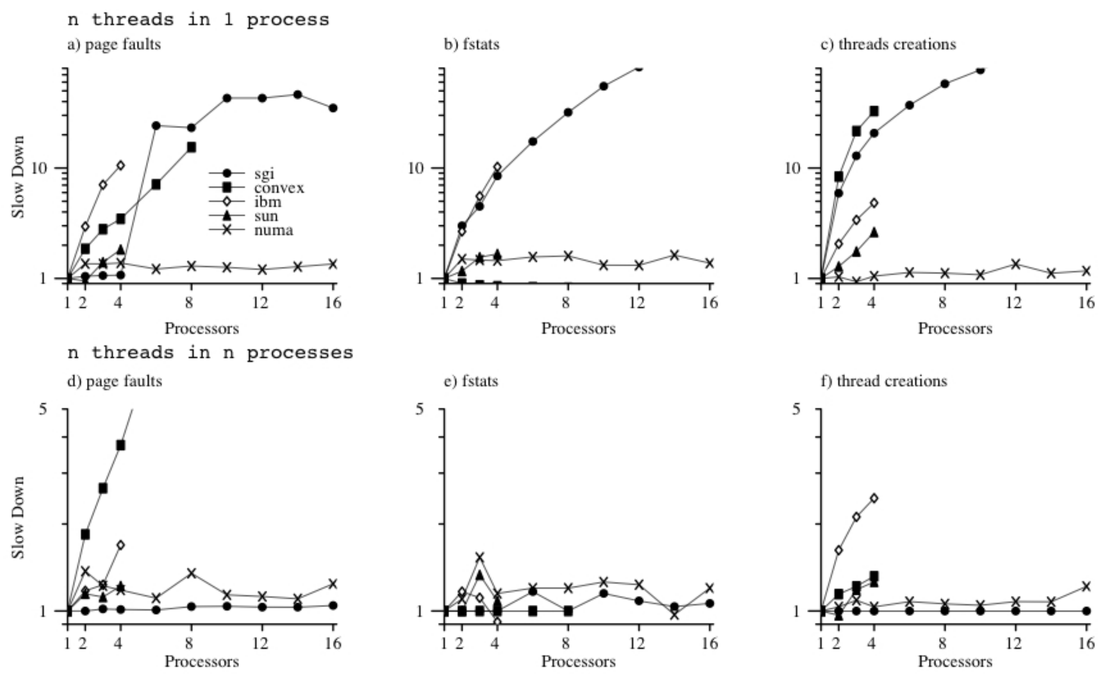
Back to Multiprocessor Scalability

- Lameter myth: IPC is needed across nodes inside a microkernel OS, and on NUMA this causes problems allocating the message queues NUMA-friendly



Whom you gonna call — local or remote OS????

Multiprocessor Scalability



- syscall slowdown vs # CPUs
- compare against several commercial systems
- only one system scales (constant slowdown)
- which is it?

What's the story?

NICTA

- **Tornado microkernel** scales perfectly to 16p
  - this is 1999! [Gamsa et al, 3rd OSDI]
  - done by a small group at Univ of Toronto
  - Tornado is predecessor of IBM's K42
- How far did Linux scale in 1999?
- How far would Linux scale **today** on the same bechmarks?
  - Note: the benchmarks show **concurrent** ops on all CPUs
    - page faults, fstats, thread creation

Synchronization Claims

- “Microkernel isolation limits synchronization methods”
- “Data structures have to be particular to subsystems”
- “Linux would never have been able to scale to these extremes with a microkernel approach because of the rigid constraints that strict microkernel designs place on the architecture of operating systems”
- **This is simply wrong (repeating doesn't make it right)**
  - synchronisation in a well-designed system is local to subsystems
  - there is no reason why subsystems can't share memory, even if microkernel-based

OS Scalability Principles

- OS must not impose synchronisation overhead except as forced by user code
- Then user code scalable ⇒ system scalable
- What does this mean?
  - keep data structures local
  - process system calls on the caller's CPU
  - only involve other CPUs if the caller explicitly asks for it!
    - creating/killing/signalling a thread on another CPU
    - invoking a synchronisation system call
    - unmap pages
- If this is done, you get a scalable OS
  - even if the apps actually perform system calls
  - user pays what user asks for...

Summary

- Hey, I can do this cool thing but you can't
  - How do you know if you don't understand me?
- Linux is cool
  - but this doesn't mean it is perfect for everything
  - nor does it mean Linux will remain as is forever
- Same is true for microkernels