

The Performance of μ -Kernel-Based Systems

by: Härtig/Hohmuth/Liedtke/Shönberg/Wolter

presented by: Eric LaForest

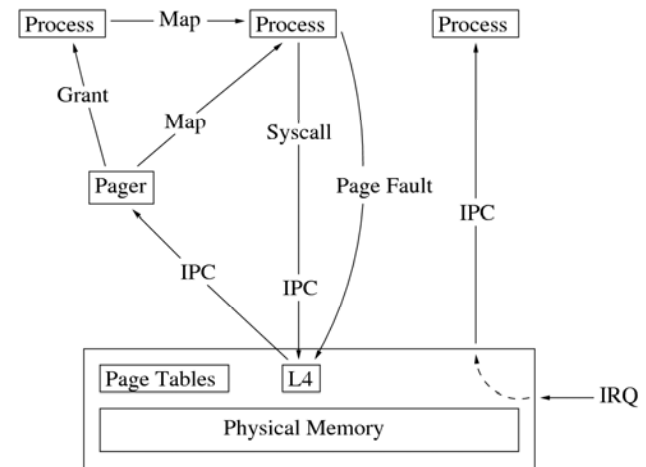
Introduction

- 1st generation of μ -kernels (monolithic-derived)
 - Chorus
 - Mach
- 2nd generation of μ -kernels (from scratch)
 - L4
 - QNX
- Evaluate with more than micro-benchmarks
 - Hence, Linux!

L4 Essentials

- Basic Concepts:
 - threads in address spaces with IPC
 - recursively so!
- Page owners can grant, map, unmap pages
- Done at virtual level, so secured by kernel
- Kernel deals with TLB, IRQ, faults, scheduling

Overview



Signal Handling

- Normally, Linux kernel signals by altering the process stack/PC, but...
- Threads in different address spaces can't touch each other, so...

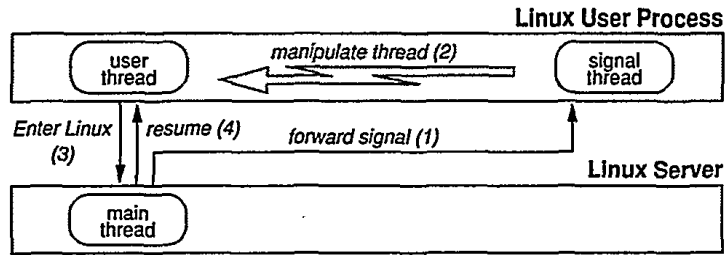


Figure 3: *Signal delivery in L⁴Linux*. Arrows denote IPC. Numbers in parentheses indicate the sequence of actions.

The Dual-Space Mistake

- Early L4 copied Linux model: user space mapped onto kernel space
- Problem: can't map multiple processes at once, switching too expensive
- Initial solution: multiple Linux server address spaces (req. synchronization)
- Actual solution: one Linux server doing software translation (reas. fast)

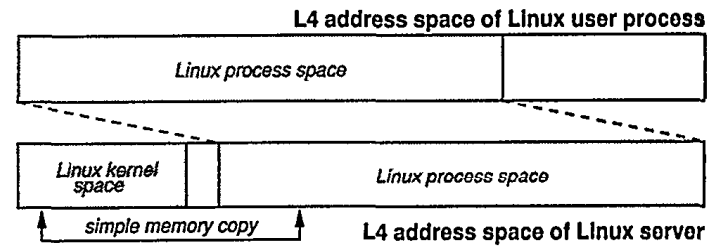
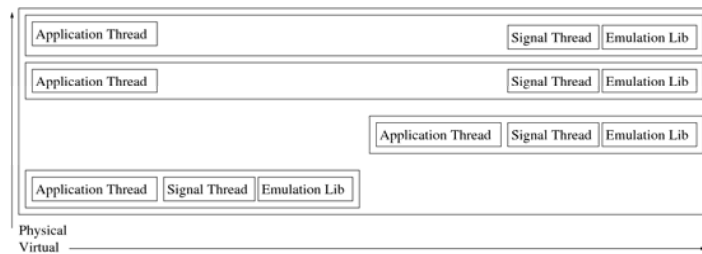


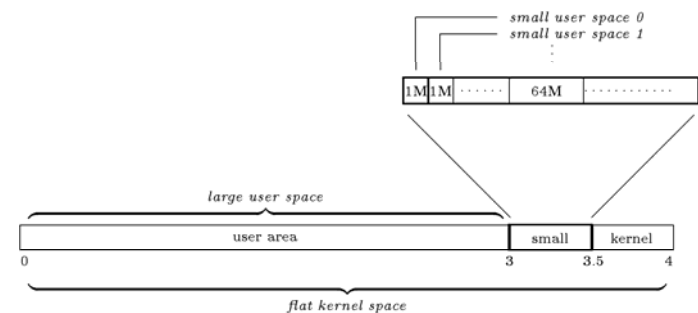
Figure 4: *Copyin/out using hardware address translation in an early version of L⁴Linux*. Arrows denote memory read/write operations,

Small Spaces

- Pentium-specific optimization to avoid TLB flushes
- Implements tagged TLB
- Improves IPC and context-switching
- Switch segment descriptor instead of page table



Small Spaces



(From Liedtke, GMD TR No.933, Nov. 1995)

System Call Overhead

System	Time	Cycles
Linux	1.68 μ s	223
L ⁴ Linux	3.95 μ s	526
L ⁴ Linux (trampoline)	5.66 μ s	753
MkLinux in-kernel	15.41 μ s	2050
MkLinux user	110.60 μ s	14710

Table 2: *getpid* system-call costs on the different implementations. (133 MHz Pentium)

Client	Cycles	Server
enter emulation library	20	
send system call message	168	wait for message
	131	— LINUX —
receive reply	188	send reply
leave emulation library	19	
	526	

Figure 5: Cycles spent for *getpid* in L⁴Linux. (133 MHz Pentium)

Macro-Benchmarks

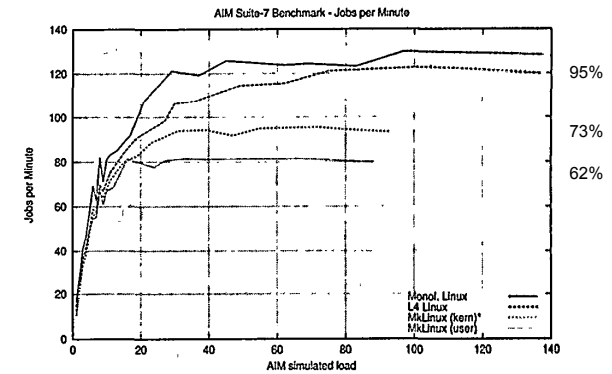


Figure 9: AIM Multiuser Benchmark Suite VII. Jobs completed per minute depending on AIM load units. (133 MHz Pentium)

Non-Unix Extensibility

- If L4Linux is no different than Linux, why use it?

Non-Unix Extensibility

- If L4Linux is no different than Linux, why use it?
- For the special cases!
 - Faster Pipes/RPC
 - Faster VM (~4x faster) with pagers
 - Cache partitioning for real-time

Non-Unix Extensibility

- If L4Linux is no different than Linux, why use it?
- For the special cases!
 - Faster Pipes/RPC
 - Faster VM (~4x faster) with pagers
 - Cache partitioning for real-time
- Anything else?

Non-Unix Extensibility

- If L4Linux is no different than Linux, why use it?
- For the special cases!
 - Faster Pipes/RPC
 - Faster VM (~4x faster) with pagers
 - Cache partitioning for real-time
- Anything else?
 - Hint: MINIX 3

Is IPC the best abstraction?

- Compared IPC to PCT (from exokernel)
- Protected Control Transfer:
 - "...a parameterless cross-address-space procedure call via a callee-defined call gate."
 - Takes the thread to a new address space

Is IPC the best abstraction?

- Compared IPC to PCT (from exokernel)
- Protected Control Transfer:
 - "...a parameterless cross-address-space procedure call via a callee-defined call gate."
 - Takes the thread to a new address space
- Pro:
 - fewer resources at target (just stacks)

Is IPC the best abstraction?

- Compared IPC to PCT (from exokernel)
- Protected Control Transfer:
 - "...a parameterless cross-address-space procedure call via a callee-defined call gate."
 - Takes the thread to a new address space
- Pro:
 - fewer resources at target (just stacks)
- Cons:
 - Lacks sync., message transfer, stack alloc.
 - does not map well to IPC, not much faster

Summary

- Based on fast IPC and efficient virtual mapping
 - more effective than in-kernel code (Mach)
- Overall performance penalty: 5-10%
- High-perf. extensions for non-Unix interfaces
- Still a live project (currently up to Linux 2.6.23):
 - <http://os.inf.tu-dresden.de/L4/LinuxOnL4/>

Summary

- Based on fast IPC and efficient virtual mapping
 - more effective than in-kernel code (Mach)
- Overall performance penalty: 5-10%
- High-perf. extensions for non-Unix interfaces
- Still a live project (currently up to Linux 2.6.23):
 - <http://os.inf.tu-dresden.de/L4/LinuxOnL4/>
- However:
 - what about residual data in mapped pages?

Thank You!

Questions?