The Performance of $\mu$-Kernel-Based Systems

H. Hartig, M. Hohmuth, J. Liedtke, S. Schonberg, and J. Wolter

Presenter: Wei-Lwun Lu
Introduction

- **Goal**
  
  Show that micro-kernel based system are usable in practice with good performance

- **Experiments**
  
  - Implement L4-Linux, and compare it with Mk-Linux and native Linux
  
  - Implement mapping related OS extension in user-level of L4-Linux
  
  - Implement L4 on an Alpha 21164 machine to check whether the L4 abstractions are reasonably independent of Pentium platform.
L4 Essentials

• Two basic concepts:
  • Threads
    • Activities executing inside an address space
    • Associated with its own pagers.
  • Address spaces
    • Recursive construction of address spaces.
    • Granting, mapping, and un-mapping
    • All address spaces are constructed and maintained by user-level servers, called pagers.
    • Page faults are propagated via IPC to the pager associated with the faulting thread. The memory management policies can be implemented in user-level.
L4 Essentials (cont.)

- I/O ports
  - treated as parts of address spaces so that can be mapped and un-mapped in the same manner as memory pages.
- Hardware interrupts
  - handled as IPC.
  - Implement all device drivers as user-level servers.
- Exceptions and traps
  - handled inside the raising threads.
- Pentium-specific feature
  - Small-address-space optimization
Linux Essentials

- Linux 2.0
- Two parts:
  - Architecture-independent: 98% of the source codes
  - Architecture-dependent: Encapsulates the underlying hardware architecture.
- Memory management uses a three-level architecture-independent page tables
- Interrupt handlers
  - Top halves: Highest priority. Can interrupt each other.
  - Bottom halves: Lower priority. Can be interrupted by the top halves, but cannot interrupt each other.
L4-Linux

- Restrict all modifications to the architecture-dependent part.
- Did not tune (optimize) Linux to L4
- Single-server approach: provide Linux services via a single Linux server.

The Linux Server

- Acts as a pager for the user process it creates.
- One thread to handle all activities induced by system calls and page faults.
- One thread per top-halve interrupt.
- One thread to handle too bottom-halve interrupt.
L4-Linux (cont.)

- Linux User Process
  - One task may have several L4 threads.
  - Each task is associated with the Linux server as its pager.

- System Calls
  - System calls are handled by IPC

- Signalling
  - L4-Linux add additional signal-handler thread to each Linux user process.

- Scheduling
  - All L4 threads are scheduled by the L4 internal scheduler.
Compatibility Performance

• Comparison
  • Native Linux 2.0
  • L4-Linux
    - Changed machine-dependent part + no optimization of Linux kernel
  • Mk-Linux
    - Changed machine-dependent part + slight optimized Linux kernel

• Micro-benchmarks
  • Measure the system-call overhead
  • Three benchmarks: getpid(), lmbench, hbench
  • L4-Linux is much more expensive than native Linux, but much cheaper than MkLinux
Micro-benchmarks

<table>
<thead>
<tr>
<th>System</th>
<th>Time</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1.68 $\mu$s</td>
<td>223</td>
</tr>
<tr>
<td>L$^4$Linux</td>
<td>3.95 $\mu$s</td>
<td>526</td>
</tr>
<tr>
<td>L$^4$Linux (trampoline)</td>
<td>5.66 $\mu$s</td>
<td>753</td>
</tr>
<tr>
<td>MkLinux in-kernel</td>
<td>15.41 $\mu$s</td>
<td>2050</td>
</tr>
<tr>
<td>MkLinux user</td>
<td>110.60 $\mu$s</td>
<td>14710</td>
</tr>
</tbody>
</table>

Table 2: `getpid` system-call costs on the different implementations. (133 MHz Pentium)
Figure 6: *imbench* results, normalized to native Linux. These are presented as slowdowns: a shorter bar is a better result. [*lat*] is a latency measurement, [*bw⁻¹*] the inverse of a bandwidth one. Hardware is a 133 MHz Pentium.
Compatibility Performance

- Macro-benchmarks
  - Measure the performance of doing real jobs
    - re-compiling the Linux server
    - AIM multiuser benchmark suite VII
  - L4-Linux was 6-7% slower than native Linux but 10-20% faster than MkLinux.
Macro-benchmarks

**Figure 7**: *Real time for compiling the Linux Server.* (133 MHz Pentium)
Macro-benchmarks (cont.)

Figure 8: *AIM Multiuser Benchmark Suite VII*. Real time per benchmark run depending on AIM load units. (133 MHz Pentium)
Figure 9: AIM Multiuser Benchmark Suite VII. Jobs completed per minute depending on AIM load units. (133 MHz Pentium)
Extensibility Performance

- Objective
  Show L4-kernel can perform better in some applications

- Three examples:
  1. Pipes and RPC
  2. Virtual Memory Operations
  3. Cache Partitioning
Extensibility Performance (cont.)

- **Pipes and RPC**
  - Implementation of synchronous L4 RPC is 5 times faster than Linux pipe with larger bandwidth.

<table>
<thead>
<tr>
<th>System</th>
<th>Latency</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Linux pipe</td>
<td>29 µs</td>
<td>41 MB/s</td>
</tr>
<tr>
<td>(1a) L⁴Linux pipe</td>
<td>46 µs</td>
<td>40 MB/s</td>
</tr>
<tr>
<td>(1b) L⁴Linux (trampoline) pipe</td>
<td>56 µs</td>
<td>38 MB/s</td>
</tr>
<tr>
<td>(1c) MkLinux (user) pipe</td>
<td>722 µs</td>
<td>10 MB/s</td>
</tr>
<tr>
<td>(1d) MkLinux (in-kernel) pipe</td>
<td>316 µs</td>
<td>13 MB/s</td>
</tr>
<tr>
<td>(2) L⁴ pipe</td>
<td>22 µs</td>
<td>48–70 MB/s</td>
</tr>
<tr>
<td>(3) synchronous L⁴ RPC</td>
<td>5 µs</td>
<td>65–105 MB/s</td>
</tr>
<tr>
<td>(4) synchronous mapping RPC</td>
<td>12 µs</td>
<td>2470–2900 MB/s</td>
</tr>
</tbody>
</table>

Table 4: *Pipe and RPC performance.* (133 MHz Pentium.) Only communication costs are measured, not the costs to generate or consume data.
Extensibility Performance (cont.)

- Virtual Memory Operations
  - Memory management can be handled in user-level and become more intelligent.
  - The resulting implementation is several times faster than the native Linux.

<table>
<thead>
<tr>
<th></th>
<th>L4</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault</td>
<td>6.2 μs</td>
<td>n/a</td>
</tr>
<tr>
<td>Trap</td>
<td>3.4 μs</td>
<td>12 μs</td>
</tr>
<tr>
<td>Appel1</td>
<td>12 μs</td>
<td>55 μs</td>
</tr>
<tr>
<td>Appel2</td>
<td>10 μs</td>
<td>44 μs</td>
</tr>
</tbody>
</table>

Table 5: *Processor time for virtual-memory benchmarks.* (133 MHz Pentium)
Extensibility Performance (cont.)

- Cache Partitioning
  - Co-existence of time-sharing and real-time memory management.
  - Use cache partitioning to reduce the worst-case execution time.
Alternative Basic Concepts

- Protected Control Transfer (PCT)
  - Implement PCT in Alpha processors
  - Compare PCT with IPC in Alpha processors:
    - PCT has no performance benefit

- Grafting
  - Compare L4 with SPIN, a sophisticated grafting technique.
  - L4 and SPIN perform roughly comparably, perhaps L4 is slightly faster.
Conclusion

- L4-Linux was 6-7% slower than native Linux but 10-20% faster than MkLinux.
- L4-kernel can perform better in some applications
Questions

- Where does the L4 microkernel improve on the Mach microkernel? What are the primary differences between the two and what in particular leads to L4's better performance? i.e., where does L4 succeed where Mach fails? And, why is Mac still using a Mach microkernel (i.e., Mach 3.0 microkernel) if its performance is so poor? Why hasn't Mac switched to L4?
Questions

- Based on its name, it would seem that L4 might be a successor to L3. But they don't seem to mention L3 at all in the paper. Does L4 have any relation to L3?
Questions

• How much does running the Linux kernel on top of L4 really tell us? It seems like there wouldn't be nearly as much IPC overhead when you have everything running in the same monolithic "kernel in userspace"
Questions

- The authors chose not to tune Linux to the microkernel [“we felt that it was beyond our means to tune Linux to our μ-kernel in the way the Mach team tuned their single-server Unix to the features of Mach.” (p3)] and made some decisions that reflect this [“multithreading at the L4 level might have been more elegant and faster. However, it would have implied a substantial change to the original Linux kernel and was thus rejected.”(p4)]. This allows for any version of Linux and any Linux program to run on L4 without much, if any, alterations. If the authors had focussed on creating the best microkernel performance-wise and didn't worry about portability, how much better do you think the performance of L4 could be? Is it worth making this decision?
Questions

• Do you think their testing approach and corresponding results really demonstrated their goals of showing that u-kernel based systems are usable in practice with good performance? Is Linux/L4 versus native Linux the best way to demonstrate this?

• Just because this L4Linux performs much better than Mach, but still falls 5-10% short (by their own admission). Are the authors valid in thinking that L4 can actually be valid based on these performance results?