The Performance of μ-Kernel-Based Systems

Hermann Härtig - Michael Hohmuth - Jochen Liedtke* - Sebastian Schönberg - Jean Wolter

Introduction.

- The Operating system community has almost abandoned research on system architecture based on pure μ-kernels.
  - Kernels that provide only address space, threads and IPC or equivalent primitive.
- The reason is poor performance reputation.

Hypothesis about μ-kernels

- Too low
  - concentrated on the extensible kernel idea.
- Too high
  - It is claimed that μ-kernels can be fast on a given architecture but can not be moved to other architecture without losing much of their efficiency.

The goal of this paper

- To show that μ-kernel based systems are usable in practice with good performance.
  - By adapting Linux to run on the top of L4, and repeating earlier experiments and conducting some novel experiments.
### Background about L4

- L4 μ-kernel based on threads and address space
  - Fast message-based synchronous IPC
  - A simple-to-use external paging-mechanism
  - A security mechanism based on secure domains
- Support recursive construction of address space by user-level servers outside the kernel.

### Background about Linux

- Linux kernel can be divided into architecture-dependent part and architecture-independent part.
- For managing address space, Linux uses a three-level architecture-independent page table scheme.
- Interrupt handlers in Linux are subdivided into top halves and bottom halves, top halves run at the highest priority.

### L4Linux Design

- Fully binary compliant with Linux/X86.
- Minimize L4-specific changes.
  - Restricted all modification to architecture-dependent part
  - No Linux-Specific modification to the L4 μ-kernel.
Linux Server

- Upon booting, the Linux server requests memory from its underlying pager.
- Then Server acts as a pager for the user processes it creates.
- The server has to keep and maintain additional logical page tables in its own address space.
- Only a single L4 thread is used for handling all activities induced by system call and page fault.

Interrupt Handling

- The L4 µ-Kernel maps hardware interrupts to messages.

Linux User Process

- Linux server create user process as a L4 task and specify itself as the associated pager.
- Linux server maps the emulation library and the signal thread code into an otherwise unused high-address part of each user address space.
System-call mechanism

- Three usable system-call interface
  - A modified version of the standard shared C library libc.so which uses L4 IPC primitive to call the Linux Server
  - A correspondingly modified version of the libc.a library
  - A user-level exception handler which emulates the native system-call trap instruction by calling a corresponding routine in the modified shared library

Signaling

- An additional signal-handler thread is added to each Linux user process
  - Native Linux kernel delivers signals to user process by directly manipulating their stack pointer and instruction pointer

Scheduling

- All threads are scheduled by the L4 µ-kernel’s internal scheduler
- Multiplexes the single Linux server thread across the multiple coroutines resulting from concurrent Linux system calls.
- Whenever a system call completes and the server’s reschedule flag is not set, the server resumes the corresponding user thread and then sleeps waiting for a new system call message or a wakeup message from interrupt handler thread.

Supporting Tagged TLBs or small space

- Tagged TLBs form the basis to avoid unnecessary TLB flushing
- For Pentium processor, small address spaces offer the possibility to emulate Tagged TLBs
- L4Linux supports small compact, application-dependent address space layout
  - A special library permitting the customization of the code and data used to communicate with Linux Server
Macrobenchmarks

AIM Multiuser Benchmark Suite VII. Real time per benchmark run depending on AIM load units. (133MHz Pentium)

11/16/99 Reported by Xiaodong Fu

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

11/16/99 Reported by Xiaodong Fu

Analysis

• What is the penalty of using L4Linux instead of native Linux?
  ■ Typical penalty range from 5% to 10%
• Does the performance of the underlying \( \mu \)-kernel matter? - yes!
• How much does co-location improve performance? - not much!
  ■ Co-located server runs in kernel mode and execute inside the \( \mu \)-kernel’s address space

Extensibility Performance

• IPC and PIPEs
  ■ we compare classical Unix pipes, pipe emulations through \( \mu \)-kernel IPC, and blocking RPC to get an estimate for the cost of emulation on various levels.

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

- Virtual Memory
  - Reimplemented the fault handlers by associating a specialized pager to the thread executing the test.

- Cache partitioning
  - Used to support real-time applications
  - A main-memory manager (a pager) on top of L4 can be used to partition the second-level cache between multiple real-time tasks and to isolate real-time from timesharing applications

Conclusion

- The performance improvements of second-generation μ-kernels significantly affect OS personalities and applications
- Fast IPC and efficient mapping abstractions are more effective than techniques such as co-location.
- In a practical scenario, the penalty for using μ-kernels can be kept somewhere between 5% and 10% for applications