The Performance of μ-Kernel Based Systems

Hermann Härtig
Michael Hohmuth
Jochen Liedtke
Sebastian Schönberg
Jean Wolter
Introduction

- $\mu$-kernels have reputation for being too slow, inflexible
- can 2nd generation $\mu$-kernel (L4) overcome limitations?
- Experiment: port Linux to run on L4 and compare it to:
  - native Linux
  - MkLinux (Linux on 1st gen Mach derived $\mu$-kernel)
Introduction

- test speed of standard OS personality on top of fast $\mu$-kernel: Linux implemented on L4
- test extensibility of system:
  - pipe-based communication implemented directly on $\mu$-kernel
  - mapping-related OS extensions implemented as user tasks
  - user-level real-time memory management implemented
- test if L4 abstractions independent of platform
L4 Essentials

- built on *threads* and *address spaces*
- recursive construction of address spaces by user-level servers
- initial address space $\sigma_0$ represents physical memory
- owner of address space can *grant* or *map* pages to another address space
- all address spaces maintained by user-level servers (*pagers*)
Linux on Top of L4

goals to keep porting effort low:

■ avoid structural changes to Linux
■ don’t tune Linux to $\mu$-kernel
■ bonus: new versions of Linux easily adapted system
L4Linux Design and Implementation

- fully binary compliant with Linux/X86
- restricted modifications to architecture-dependent part of Linux
- no Linux-specific modifications to L4 kernel
- single Linux server task
- Linux server requests memory from $\sigma_0$, acts as pager for user processes
Figure 1: $L^4\text{Linux Address Spaces}$. Arrows denote mapping. The Linux server space can be a subset of $\sigma_0$. Although plotted as smaller boxes, the user address spaces can be larger than the server’s address space.
L4Linux Design and Implementation

- L4 maps hardware interrupts to messages
- Linux top-half handlers implemented as threads waiting for messages
3 system call interfaces:

- modified version of standard libc.so which uses L4 IPC
- modified version of standard libc.a which uses L4 IPC
- “trampoline” which emulates native system-call trap (slower, but full binary compatibility)
L4Linux Design and Implementation

- 6500 new lines of code written
- 14 engineer-months to build L4Linux
- regularly use applications such as X Windows, Emacs, Netscape, X-Pilot
Compatibility Performance

- What is penalty of using L4Linux instead of native Linux?
- Does performance of underlying $\mu$-kernel matter?
- How much does co-location improve performance?
- microbenchmarks: analyze detailed behavior
- macrobenchmarks: measure overall system performance

The Performance of $\mu$-Kernel Based Systems – p. 11
Microbenchmarks

- measure system call overhead on shortest system call (\texttt{getpid()})

<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: \texttt{getpid} system-call costs on the different implementations. (133 MHz Pentium)
Microbenchmarks

- *lmbench* measures basic operations like system calls, context switch, pipe operations, network operation.

![Graph of lmbench results, normalized to native Linux.](image)

**Figure 6. lmbench results, normalized to native Linux.** These are presented as slowdowns: a shorter bar is a better result. [lat] is latency measurement, [bw⁻¹] the inverse of a bandwidth one. Hardware is a 133 MHz Pentium.
Macrobenchmarks

- measured time to recompile Linux server

<table>
<thead>
<tr>
<th>System</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>476</td>
</tr>
<tr>
<td>L^4Linux</td>
<td>506 (+6.3%)</td>
</tr>
<tr>
<td>L^4Linux (trampo)</td>
<td>509 (+6.9%)</td>
</tr>
<tr>
<td>MkLinux (kernel)</td>
<td>555 (+16.6%)</td>
</tr>
<tr>
<td>MkLinux (user)</td>
<td>605 (+27.1%)</td>
</tr>
</tbody>
</table>

*Figure 7: Real time for compiling the Linux Server. (133 MHz Pentium)*
Macrobenchmarks

- AIM multiuser benchmark VII (measures system performance under different loads)

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)
Analysis

- L4 comes close to performance of native Linux, even under high load (5%-10%)
- Performance of underlying $\mu$-kernel matters
- Co-location is not sufficient to overcome performance deficiencies