µ-Kernels

CS6410 September 20, 2011
Patrick Dowell
What is a Kernel?

- Helps provide a layer of abstraction between applications and hardware
- Lowest layer of abstraction in OS
Types of Kernels

- Monolithic Kernels
- $\mu$-Kernels
- Hybrid Kernels
- Nanokernels
- Exokernels
Monolithic Kernels

- All OS code runs in kernel space
- Same level of privilege
- Same address space
- Kernel code becomes large, difficult to code and debug
- One bug can bring the whole thing down
- Ex: Multics, Unix, Linux, DOS, Windows 9x
Monolithic Kernels
Kernel Responsibilities

- Scheduling
- Memory Management
- IPC
- Device I/O (including file systems)
- Security (including CPU mode)
- User Interface
\(\mu\)-Kernels

- Minimum software needed for an OS
- Kernel includes scheduling, memory management, and IPC
- Everything else in User Space
- Examples: Mach, L4, seL4
Kernel Comparison
μ-Kernels Pros & Cons

• Clean system architecture
• Easy to maintain
• Better fault tolerance of components
• Better portability
• More memory needed
• Performance loss due to context switches
The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System

- SOSP - 1987

- Notable Authors:
  - Richard Rashid - Lead Developer of Mach, Senior Vice President of Microsoft Research since 2000
  - Avadis Tevanlan - Former Head of Software at NeXT, Chief Software Tech Officer at Apple until 2006
Mach

- One of the first µ-kernels
- Developed at CMU from 1985-1994
- Successor to CMU’s Accent kernel
- v2.5 integrated into the XNU kernel, appears in NeXTSTEP, Mac OS X, and iOS.
Mach Primitives

- **task** - basic unit of resource allocation
- **thread** - basic unit of computation
- **port** - communication channel (many senders, one receiver)
- **message** - variable size collection of objects
Inter-Process Communication

- All data accessed by messages
- Ports serve as capabilities to objects
- Messages can transfer task’s address space
- Upon creation of task/thread, creator is given port to send messages
- Integrates with networked systems
Virtual Memory

• Task address space is ordered collection of memory regions

• Tasks create ports for their objects

• Copy-on-Write sharing used during task creation and message passing

• VM serves as a cache for Memory Objects
External Memory

- All data in secondary storage represented by a Memory Object
- Ports serve as capabilities to MOs
- Mach acts as cache manager for MOs
- Modified data in VM must be flushed
- Asynchronous RPC used for Read/Write
Applications

- UNIX
- Process migration via copy-on-reference
- Camelot - distributed transactions
- Agora - distributed speech recognition
Problems

- Use of IPC for external storage introduced timeouts, complexity to file reads / writes
- Ports and services are difficult to locate
- While User Space calls are cheap, Kernel Space calls are frequent & expensive
Performance?

• No detailed evaluation
• Huge performance hit due to heavy IPCs
• We’ll see numbers in the next paper
The Performance of \(\mu\)-Kernel Based Systems

- SOSP - 1997
- Notable Authors:
  - Jochen Leidtche - Worked on ELAN, Eurnel, L3, and L4
  - Demonstrated that \(\mu\)-Kernels performed well if designed properly
L4

• Second generation µ-Kernel
• Developed at IBM Watson Research Center
• Widely used, formed the basis for many successors
• Used to create highly secure & reliable systems
L4 Primitives

- **Thread** - activity inside an address space
- **Address Space** - constructed atop physical memory, $\sigma_0$
- **IPC** - communicate across address spaces
- **Pagers** - user level servers recursively map address spaces on top $\sigma_0$
- **I/O ports** - treated like memory pages
L4Linux Essentials

- Linux System calls are RPCs - IPCs between User process and Linux server
- Linux server operates in $\sigma_0$
- One thread used for sys calls & page faults
- Trampolines handle exceptions in user space
- Tagged TLBs avoid unnecessary CPU cache flushes for small address spaces
L4 Innovations

- Lightweight IPC used with low overhead
- Optimized heavily in assembly
- No rights management in μ-kernel
- First μ-kernel to show high performance
Evaluation

• Comparison of Linux, MkLinux, & L4Linux

• Goals:
  • find overhead of L4Linux over Linux
  • evaluate improvement of L4 over Mach
  • show IPC significantly faster in L4Linux
getpid system call

<table>
<thead>
<tr>
<th>System</th>
<th>Time</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1.68 μs</td>
<td>223</td>
</tr>
<tr>
<td>L⁴Linux</td>
<td>3.95 μs</td>
<td>526</td>
</tr>
<tr>
<td>L⁴Linux (trampoline)</td>
<td>5.66 μs</td>
<td>753</td>
</tr>
<tr>
<td>MkLinux in-kernel</td>
<td>15.41 μs</td>
<td>2050</td>
</tr>
<tr>
<td>MkLinux user</td>
<td>110.60 μs</td>
<td>14710</td>
</tr>
</tbody>
</table>

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

L⁴Linux is 2.4x slower than Linux here!
Imbench Results

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.

1.5-2.5x slowdown seen here for L4Linux!
AIM Benchmark

Figure 8: AIM Multiuser Benchmark Suite VII. Real time per benchmark run depending on AIM load units. (133 MHz Pentium)

Over all loads, L4Linux is 8.3% slower than Linux
Pipes/RPC Performance

<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.

Using sync L4 RPC yields a 6x improvement!
Hybrid Kernels

- Scheduling, VM, IPC, and device drivers handled by kernel
- Low performance overhead for messaging, context switching
- No performance benefit to services in user space
- Ex: Win NT, 2000, XP, Mac OS X, iOS
Conclusion

• µ-kernels are example of end-to-end argument for kernels
• With Mach, clean architecture, poor performance
• L4 shows that µ-kernels can perform well
• Hybrid kernels combine best features of monolithic and µ-kernels