Critique Of Microkernel Architectures

I’m not interested in making devices look like user-level. They aren’t, they shouldn’t, and microkernels are just stupid.  
*Linus Torvalds*

Is Linus right?
Microkernel Performance

• First generation μ-kernel systems exhibited poor performance when compared to monolithic UNIX implementations.
  – particularly Mach, the best-known example
• Reasons are investigated by [Chen & Bershad, 1993]:
  – instrumented user and system code to collect execution traces
  – run on DECstation 5000/200 (25MHz R3000)
  – run under Ultrix and Mach with Unix server
  – traces fed to memory system simulator
  – analyse MCPI (memory cycles per instruction)
  – baseline MCPI (i.e. excluding idle loops)
Interpretation

Observations:
• Mach memory penalty (i.e. cache misses or write stalls) higher
• Mach VM system executes more instructions than Ultrix (but has more functionality).

Claim:
• Degraded performance is (intrinsic?) result of OS structure.
• IPC cost (known to be high in Mach) is not a major factor [Bershad, 1992].
Assertions

1. OS has less instruction and data locality than user code.
   - System code has higher cache and TLB miss rates.
   - Particularly bad for instructions.
Assertions

2. System execution is more dependent on instruction cache behaviour than is user execution
   – MCPIs dominated by system i-cache misses.
   – Note: most benchmarks were small, i.e. user code fits in cache.

<table>
<thead>
<tr>
<th>workload</th>
<th>instruction cache</th>
<th>data cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ultrix</td>
<td>Mach</td>
</tr>
<tr>
<td></td>
<td>sys   user</td>
<td>sys   user</td>
</tr>
<tr>
<td>sed</td>
<td>0.129 0.005</td>
<td>0.283 0.005</td>
</tr>
<tr>
<td>egrep</td>
<td>0.014 0.001</td>
<td>0.046 0.001</td>
</tr>
<tr>
<td>yacc</td>
<td>0.028 0.004</td>
<td>0.069 0.003</td>
</tr>
<tr>
<td>gcc</td>
<td>0.103 0.145</td>
<td>0.294 0.123</td>
</tr>
<tr>
<td>compress</td>
<td>0.060 0.002</td>
<td>0.157 0.005</td>
</tr>
<tr>
<td>ab</td>
<td>0.139 0.130</td>
<td>0.261 0.098</td>
</tr>
<tr>
<td>espresso</td>
<td>0.009 0.012</td>
<td>0.026 0.011</td>
</tr>
<tr>
<td>lisp</td>
<td>0.002 0.001</td>
<td>0.013 0.011</td>
</tr>
<tr>
<td>eqntott</td>
<td>0.001 0.000</td>
<td>0.003 0.000</td>
</tr>
<tr>
<td>fpppp</td>
<td>0.050 0.184</td>
<td>0.040 0.173</td>
</tr>
<tr>
<td>dduc</td>
<td>0.014 0.277</td>
<td>0.020 0.270</td>
</tr>
<tr>
<td>liv</td>
<td>0.013 0.000</td>
<td>0.045 0.000</td>
</tr>
<tr>
<td>tomcatv</td>
<td>0.000 0.000</td>
<td>0.002 0.000</td>
</tr>
</tbody>
</table>
Assertions

3. Competition between user and system code is not a problem
   - Few conflicts between user and system caching.
   - TLB misses are not a relevant factor.
   - Note: the hardware used has direct-mapped physical caches.

$\Rightarrow$ Split system/user caches wouldn't help.
Self-Interference

- Only examine system cache misses.
- Shaded: System cache misses removed by associativity.
- MCPI for system-only, using R3000 direct-mapped cache.
- Reductions due to associativity were obtained by running system on a simulator and using a two-way associative cache of the same size.
Assertions…

4. **Self-interference is a problem in system instruction reference streams.**
   - High internal conflicts in system code.
   - System would benefit from higher cache associativity.

5. **System block memory operations are responsible for a large percentage of memory system reference costs.**
   - Particularly true for I/O system calls.

6. **Write buffers are less effective for system references.**
   - Write buffer allows limited asynch. writes on cache misses.

7. **Virtual to physical mapping strategy can have significant impact on cache performance**
   - Unfortunate mapping may increase conflict misses.
   - “Random” mappings (Mach) less likely to exhibit consistently poor performance.
Other Experience With Microkernel Performance

- System call costs are (inherently?) high.
  - Typically hundreds of cycles, 900 for Mach/i486.
- Context (address-space) switching costs are (inherently?) high.
  - Getting worse (in terms of cycles) with increasing CPU/memory speed ratios [Ousterhout, 1990].
  - IPC (involving system calls and context switches) is inherently expensive.
So, What’s Wrong?

- The MCPI for Mach is significantly higher than Ultrix
- \(\mu\)-kernels heavily depend on IPC
- IPC is expensive
  - Is the \(\mu\)-kernel idea flawed?
  - Should some code never leave the kernel?
  - Do we have to buy flexibility with performance?
A Critique Of The Critique

- Data presented earlier:
  - are specific to one (or a few) system,
  - results cannot be generalised without thorough analysis,
  - no such analysis has been done.

⇒ Cannot trust the conclusions [Liedkte, 1995].
Re-analysis Of Chen & Bershad’s Data

MCPI for Ultrix and Mach
Re-analysis Of Chen & Bershad’s Data...

<table>
<thead>
<tr>
<th>Program</th>
<th>Conflict Misses</th>
<th>Capacity Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>sed</td>
<td>0.170</td>
<td>0.415</td>
</tr>
<tr>
<td>egrep</td>
<td>0.024</td>
<td>0.069</td>
</tr>
<tr>
<td>yacc</td>
<td>0.039</td>
<td>0.098</td>
</tr>
<tr>
<td>gcc</td>
<td>0.130</td>
<td>0.388</td>
</tr>
<tr>
<td>compress</td>
<td>0.102</td>
<td>0.258</td>
</tr>
<tr>
<td>ab</td>
<td>0.230</td>
<td>0.382</td>
</tr>
<tr>
<td>espresso</td>
<td>0.012</td>
<td>0.037</td>
</tr>
</tbody>
</table>

MCPI caused by cache misses: conflict (black) vs capacity (white)
Conclusion

- Mach system (kernel + UNIX server + emulation library) is too big!
- UNIX server is essentially same.
- Emulation library is irrelevant (according to Chan & Bershad).

⇒ Mach μ-kernel working set is too big

Can we build μ-kernels which avoid these problems?
Requirements For Microkernels:

- Fast (system call costs, IPC costs)
- Small (big $\Rightarrow$ slow)

$\Rightarrow$ Must be well designed, providing a minimal set of operations.

Can this be done?
Are High System Costs Essential?

- Example: kernel call cost on i486
  - Mach kernel call: 900 cycles
  - Inherent (hardware-dictated cost): 107 cycles.
    \[ \Rightarrow \text{800 cycles kernel overhead.} \]
    \[ \Rightarrow \text{Mach’s performance is a result of design and implementation not the } \mu\text{-kernel concept!} \]
Microkernel Design Principles (Liedtke)

- **Minimality**: If it doesn’t *have to be* in the kernel, it *shouldn’t* be in the kernel
  - Security is the only case for *must be in the kernel*
- **Appropriate abstractions** which can be made fast and allow efficient implementation of services
- **Well written**: It pays to shave a few cycles off TLB refill handler or the IPC path
- **Unportable**: must be targeted to specific hardware
  - no problem if it’s small, and higher layers are portable
  - Example: Liedtke reports significant rewrite of memory management when porting from 486 to Pentium
    ⇒ “abstract hardware layer” is too costly
NON-PORTABILITY EXAMPLE: I486 VS PENTIUM:

- Size and associativity of TLB
- Size and organisation of cache (larger line size - restructured IPC)
- Segment regs in Pentium used to simulate tagged TLB

⇒ different trade-offs
WHAT must A µ-KERNEL PROVIDE?

- Virtual memory/address spaces
- threads,
- fast IPC,
- unique identifiers (for IPC addressing).

µ-KERNEL DOES not HAVE TO PROVIDE:

- file system
  - use user-level server (as in Mach)
- device drivers
  - user-level driver invoked via interrupt (= IPC)
- page-fault handler
  - use user-level pager
L4 Implementation Techniques

- Appropriate system calls to reduce number of kernel invocations
  - e.g., reply & receive next
- Rich message structure
  - value and reference parameters in message
- Copy message only once (i.e. not user!kernel!user)
- Short messages in registers
- As many syscall parameters in registers as possible
- One kernel stack (for interrupt handling) per thread (in TCB)
- TCBs in (mapped) VM, cache-friendly layout
- Thread UIDs (containing thread ID)
- “Hottest” kernel code is shortest
- Kernel IPC code on single page, critical data on single page
- Many H/W specific optimisations
## Performance

<table>
<thead>
<tr>
<th>System</th>
<th>CPU</th>
<th>MHz</th>
<th>RPC $\mu$s</th>
<th>cyc/IPC</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4</td>
<td>R4600</td>
<td>100</td>
<td>1.7 $\mu$s</td>
<td>100</td>
<td>full</td>
</tr>
<tr>
<td>L4</td>
<td>Alpha</td>
<td>433</td>
<td>0.2 $\mu$s</td>
<td>45</td>
<td>full</td>
</tr>
<tr>
<td>L4</td>
<td>Pentium</td>
<td>166</td>
<td>1.5 $\mu$s</td>
<td>121</td>
<td>full</td>
</tr>
<tr>
<td>L4</td>
<td>486</td>
<td>50</td>
<td>10 $\mu$s</td>
<td>250</td>
<td>full</td>
</tr>
<tr>
<td>QNX</td>
<td>486</td>
<td>33</td>
<td>76 $\mu$s</td>
<td>1254</td>
<td>full</td>
</tr>
<tr>
<td>Mach</td>
<td>R2000</td>
<td>16.7</td>
<td>190 $\mu$s</td>
<td>1584</td>
<td>full</td>
</tr>
<tr>
<td>SCR RPC</td>
<td>CVAX</td>
<td>12.5</td>
<td>464 $\mu$s</td>
<td>2900</td>
<td>full</td>
</tr>
<tr>
<td>Mach</td>
<td>486</td>
<td>50</td>
<td>230 $\mu$s</td>
<td>5750</td>
<td>full</td>
</tr>
<tr>
<td>Amoeba</td>
<td>68020</td>
<td>15</td>
<td>800 $\mu$s</td>
<td>6000</td>
<td>full</td>
</tr>
<tr>
<td>Spin</td>
<td>Alpha 21064</td>
<td>133</td>
<td>102 $\mu$s</td>
<td>6783</td>
<td>full</td>
</tr>
<tr>
<td>Mach</td>
<td>Alpha 21064</td>
<td>133</td>
<td>104 $\mu$s</td>
<td>6916</td>
<td>full</td>
</tr>
<tr>
<td>Exo-trlpc</td>
<td>R2000</td>
<td>116.7</td>
<td>6 $\mu$s</td>
<td>53</td>
<td>restricted</td>
</tr>
<tr>
<td>Spring</td>
<td>SparcV8</td>
<td>40</td>
<td>11 $\mu$s</td>
<td>220</td>
<td>restricted</td>
</tr>
<tr>
<td>DP-Mach</td>
<td>486</td>
<td>66</td>
<td>16 $\mu$s</td>
<td>528</td>
<td>restricted</td>
</tr>
<tr>
<td>LRPC</td>
<td>CVAX</td>
<td>12.5</td>
<td>157 $\mu$s</td>
<td>981</td>
<td>restricted</td>
</tr>
</tbody>
</table>
Case In Point: L4Linux
[Härtig et al., 1997]

• Port of Linux kernel to L4 (like Mach Unix server)
  – single-threaded (for simplicity, not performance)
  – is pager of all Linux user processes
  – maps emulation library and signal-handling code into AS
  – server AS maps physical memory (& Linux runs within)
  – copying between user and server done on physical memory
  – use software lookup of page tables for address translation

• Changes to Linux restricted to architecture-dependent part
• Duplication of page tables (L4 and Linux server)
• Binary compatible to native Linux via trampoline mechanism
  – but also modified libc with RPC stubs
L⁴Linux Overview
Server Internals

- L4 threads used to
  - receive device interrupts
  - Emulated Linux’s *bottom half* handling
  - Receive system calls from applications
Signal Delivery In L⁴Linux

• Separate signal-handler thread in each user process
  - server IPCs signal-handler thread
  - handler thread ex regs main user thread to save state
  - user thread IPCs Linux server
  - server does signal processing
  - server IPCs user thread to resume
L$^4$Linux Performance

Microbenchmarks:

<table>
<thead>
<tr>
<th>System</th>
<th>Time [μs]</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>1.68</td>
<td>223</td>
</tr>
<tr>
<td>L$^4$Linux</td>
<td>3.95</td>
<td>526</td>
</tr>
<tr>
<td>L$^4$Linux (trampoline)</td>
<td>5.66</td>
<td>753</td>
</tr>
<tr>
<td>MkLinux in-kernel</td>
<td>15.66</td>
<td>2050</td>
</tr>
<tr>
<td>MkLinux server</td>
<td>110.60</td>
<td>14710</td>
</tr>
</tbody>
</table>

getpid() on 133MHz Pentium
Cycle Breakdown

<table>
<thead>
<tr>
<th>Client</th>
<th>Cycles</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>enter emulation lib</td>
<td>20</td>
<td>wait for msg</td>
</tr>
<tr>
<td>send syscall message</td>
<td>168</td>
<td>Linux kernel</td>
</tr>
<tr>
<td>receive reply</td>
<td>131</td>
<td>send reply</td>
</tr>
<tr>
<td>leave emulation lib</td>
<td>188</td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

Hardware cost: 82 cycles
Macrobenchmarks: LMBENCH

- write /dev/null [lat]
- null process [lat]
- simple process [lat]
- /bin/sh process [lat]
- mmap [lat]
- 2-proc context switch [lat]
- 8-proc context switch [lat]
- pipe [lat]
- UDP [lat]
- RPC/UDP [lat]
- TCP [lat]
- RPC/TCP [lat]
- pipe [bw-l]
- TCP [bw-l]
- file reread [bw-l]
- mmap reread [bw-l]

- Linux
- L^2Linux
- MkLinux (in-kernel)
- MkLinux (user)

- 64.5
- 25.8
# Macrobenchmarks: Kernel Compile

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

• Mach sux ≠ microkernels suck
• L4 shows that performance *might* be deliverable
  – L⁴Linux gets close to monolithic kernel performance
  – need real multi-server system to evaluate μ-kernel potential
• Jury is still out!
• Mach has prejudiced community (see Linus...)
  – It’ll be an uphill battle!
## Implementations

<table>
<thead>
<tr>
<th>API</th>
<th>Kernel</th>
<th>Who</th>
<th>Language</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>L4/x86</td>
<td>Liedtke</td>
<td>asm</td>
<td>x86</td>
</tr>
<tr>
<td></td>
<td>L4/MIPS</td>
<td>UNSW</td>
<td>asm/C</td>
<td>R4k</td>
</tr>
<tr>
<td></td>
<td>L4/Alpha</td>
<td>UNSW/Dres</td>
<td>PAL/C</td>
<td>21x64</td>
</tr>
<tr>
<td></td>
<td>Fiasco</td>
<td>Dresden</td>
<td>C++</td>
<td>x86</td>
</tr>
<tr>
<td>X.0</td>
<td>L4/x86</td>
<td>Liedtke</td>
<td>asm</td>
<td>x86</td>
</tr>
<tr>
<td></td>
<td>Hazelnut</td>
<td>Karlsruhe</td>
<td>C</td>
<td>x86, ARM</td>
</tr>
<tr>
<td>V4</td>
<td>Pistachio</td>
<td>Karlsruhe</td>
<td>C++</td>
<td>x86, IA-64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UNSW</td>
<td></td>
<td>PPC-32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MIPS, Alpha</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ARM, PPC-64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SPARC (i.p.)</td>
</tr>
</tbody>
</table>