"Improving IPC by Kernel Design"
(J. Liedtke, et.al.)

Indradip Ghosh
(CS 533, Winter 2012)
Main Idea

- Mach OS being a first generation μ-Kernel, had a bunch of performance smells
- Liedtke identified 17 counts of Optimizations on 4 subcategories based on Kernel design:
  - Architectural level
  - Algorithmic level
  - Interface level
  - Coding level
- Result: L3 μ-Kernel is Upto 22 times faster than Mach
L3 Kernel Design Philosophy

- **Design in favor of performance**
  - Every design decision must have a performance validation
  - Any technique that has performance issues has to be reworked

- **Holistic system view**
  - A newly added feature has to be performance and functionally validated against the whole system and not as stand alone
  - Design has to cover all levels: Architecture to Coding

- **Concrete Performance Goal**
  - Needs a statement of assumptions about hardware support
    - e.g., L3 is built on 486 MMU capabilities (4-way set-associative flushable TLB and 32 bit virtual addressing & 2 level Page Tables)
  - What to benchmark against (T = 5μs for short message transfer)
To get an idea of what’s costly?

At least 5 TLB Misses for a NULL IPC

<table>
<thead>
<tr>
<th>thread A (user mode):</th>
</tr>
</thead>
<tbody>
<tr>
<td>load id of B</td>
</tr>
<tr>
<td>set msg length to 0</td>
</tr>
<tr>
<td>call kernel</td>
</tr>
<tr>
<td>access B</td>
</tr>
<tr>
<td>load id of A</td>
</tr>
<tr>
<td>switch stack space</td>
</tr>
<tr>
<td>load id of A</td>
</tr>
<tr>
<td>return to user</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel:</th>
</tr>
</thead>
<tbody>
<tr>
<td>access thread B</td>
</tr>
<tr>
<td>switch stack pointer</td>
</tr>
<tr>
<td>switch address space</td>
</tr>
<tr>
<td>load id of A</td>
</tr>
<tr>
<td>return to user</td>
</tr>
<tr>
<td>inspect received msg</td>
</tr>
</tbody>
</table>

Determining minimum IPC cost

✓ 0 length messages between address spaces
✓ receiver is ready to receive it
✓ 127 Execution cycles; 107 for kernel-user mode switch
✓ 45 cycles for TLB misses
✓ Total CPU cycles = 172 (127 + 45)
Reduce Protection Domain Transitions

- Minimize System Calls
  - System calls cost 60% T
  - Traditional IPC requires 4 system calls

- Solution
  - Synchronous call semantics permits 2 system calls instead
  - IPC client calls into kernel and blocks on \texttt{receive(reply)}
  - IPC server calls into kernel to send reply and blocks on \texttt{receive(next)}
Two system call based IPC
Combine complex messages and send them in 1 system call

A Complex Message

Direct String
  Copy of data to be transferred in the message

Indirect String
  Address and length of data to be transferred in the message

Memory Object
  Description of a region of memory to be mapped in receiver
  address space (such as a file)
Message Transfer Mechanisms

- Traditional systems make a two fold copy:
- IPC client user space $\rightarrow$ Kernel Space $\rightarrow$ IPC server user space

- LRPC mechanism uses shared user space
- Dubious security, violates protection domain semantics
- Also not application friendly, w.r.t fine grained messaging
Temporary Mapping

- Two copy costs $20 + 0.75 \times n$ cycles + TLB misses
- L3 uses special communication window in kernel space for single copy message transfer
- Window is mapped to the receiver for the duration of the call.....How?
- Is Secure.....Why ?
Virtual Memory Abstraction

VM maps CPU generated 32/64 bit virtual addresses to physical memory page addresses, that could be resident or non-resident in main memory by a combination of hardware (TLB) and software (Page Tables) artifacts.

**TLB**: hardware cache of virtual addresses from the page directory (pdir) in the main memory

**VM** provides the abstraction that the physical memory is as big as the processor address space and also automates the translation of these large virtual addresses to physical addresses for programming convenience.
Virtual Address Translation from the TLB to frames in physical memory is done using Two level page tables.
Fast Temporary Mapping

- Only requires copying of one word of the page directory from caller to server
- Message strings are restricted to $2^{12}$ for fast mapping to work correctly
- For the Communication Window to work correctly the TLB has to be flushed each time there is a thread switch … Why?
Lazy scheduling

- Conventionally, IPC call requires 2 ENQ & 2 DEQ operations in a single IPC as the caller and callee threads block/unblock.
- This incurs an overhead of 23% T.
- Instead, the thread state variable of the thread in the TCB is changed from ready to waiting and inserted at the end of the ready Q.
- This removes work being done in the busy IPC path and adding a little bit of work in a much infrequent timer interrupt path.
- IPC: lazy Q update operations can lead up to 50:1 ratio.
Short Messages via Registers

- **Optimization scope**
  - A high proportion of messages are very short, e.g., hardware initiated interrupt messages
  - 50% to 80% of messages leq 8 bytes
  - 486 has 7 general purpose registers

- **Result**
  - Direct Transfer incurred overhead 25% T
  - Register based transfer had a Perf. gain 48% T
Summary of Techniques

- **Architectural**
  - System Calls, Messages, Direct Transfer, Strict Process Orientation, Thread Control Blocks

- **Algorithmic**
  - Thread Identifier, Virtual Queues, Timeouts/Wakeups, Lazy Scheduling, Direct Process Switch, Short messages

- **Interface**
  - Unnecessary Copies, Parameter passing

- **Coding**
  - Cache misses, TLB misses, Segment registers, General registers, Jumps and Checks, Process Switch
# Results

<table>
<thead>
<tr>
<th>removed optimization</th>
<th>time increase (n-byte ipc)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>short msg via reg</td>
<td>49%</td>
</tr>
<tr>
<td>direct transfer</td>
<td></td>
</tr>
<tr>
<td>lazy scheduling</td>
<td>23%</td>
</tr>
<tr>
<td>no segm reg</td>
<td>21%</td>
</tr>
<tr>
<td>reply &amp; wait(^a)</td>
<td>18%</td>
</tr>
<tr>
<td>condensed tables(^b)</td>
<td>13%</td>
</tr>
<tr>
<td>virtual tcb(^c)</td>
<td>10%</td>
</tr>
</tbody>
</table>

\(^a\)when used to implement RPC  
\(^b\)due to less TLB misses  
\(^c\)only reduced TLB miss effect

## Table 1: Easily quantifiable Effects
Results

L3's message passing is up to 22 times faster than that of MACH.
Conclusions

- Fast IPC is possible performance based reasoning at all levels of the system
- The techniques are generic enough to be applied to other hardware and µ-kernel systems with a few exceptions
- Fact to boot though, why would people use optimization unfriendly hardware