Micro-kernel
Key points

• Microkernel provides minimal abstractions
  – Address space, threads, IPC

• Abstractions
  – … are machine independent
  – But *implementation* is hardware dependent

• Myths about inefficiency of micro-kernel stem from inefficient implementation and not from microkernel approach
What abstractions?

• Determining criterion:
  – Functionality not performance

• Hardware and microkernel should be trusted but applications are not
  – Hardware provides page-based virtual memory
  – Kernel builds on this to provide protection for services above and outside the microkernel

• Principles of independence and integrity
  – Subsystems independent of one another
  – Integrity of channels between subsystems protected from other subsystems
Microkernel Concepts

• Hardware mapping provides address space
  – from virtual page to a physical page
  – implemented by page tables and TLB

• Microkernel concept of address spaces
  – Hides the hardware address spaces and provides an abstraction that supports
    • Grant?
    • Map?
    • Flush?
  – These primitives allows building a hierarchy of protected address spaces
Grant, Map and Flush

QuickTime™ and a decompressor are needed to see this picture.
• Power and flexibility of address spaces
  – Initial memory manager for address space A0 appears by magic and encompasses the physical memory
  – Allow creation of stackable memory managers (all outside the kernel)
  – Pagers can be part of a memory manager or outside the memory manager
  – All address space changes (map, grant, flush) orchestrated via kernel for protection
  – Device driver can be implemented as a special memory manager outside the kernel as well
Microkernel

processor

Map/grant

M0, A0, P0

PT

M1, A1, P1

PT

M2, A2, P2

PT
Threads and IPC

- A thread executes in an address space
  - Associated (or represented by) PC, SP, processor registers, and state info (such as address space)
- IPC is cross-address-space communication
  - Supported by the microkernel
    - Classic method is message passing between threads via the kernel
    - Sender sends info; receiver decides if it wants to receive it, and if so where
    - Address space operations such as map, grant, flush need IPC
  - Higher level communication (e.g. RPC) built on top of basic IPC
• Interrupts?
  – Each hardware device is a thread from kernel’s perspective
  – Interrupt is a null message from a hardware thread to the software thread
  – Kernel transforms hardware interrupt into a message
    • Does not know or care about the semantics of the interrupt
    • Device specific interrupt handling outside the kernel
    • Clearing hardware state (if privileged) then carried out by the kernel upon driver thread’s next IPC

• TLB handler?
  • In theory software TLB handler can be outside the microkernel
  • In practice first level TLB handler inside the microkernel or in hardware
Unique IDs

- Kernel provides uid over space and time for
  - Threads
  - IPC channels
Breaking some performance myths

• Kernel-user switches
• Address space switches
• Thread switches and IPC
• Memory effects

Base system:
  486 (50 MHz) – 20 ns cycle time
Kernel-user switches

• Machine instruction for entering and exiting
  – 107 cycles
  – Mach measures 900 cycles for kernel-user switch
    • Why?
  – Empirical proof
    • L3 kernel ~ 123 cycles (accounting for some TLB, cache misses)
  – Where did the remaining 800 cycles go in MACH?
    • Kernel overhead (construction of the kernel, and inherent in the approach)
Address space switches

• TLBs

• Instruction and data caches
  – Usually physically tagged in most modern processors so TLB flush has no effect

• Address space switch
  – Complete reload of Pentium TLB ~ 864 cycles
• Do we need a TLB flush always?
  – Implementation issue of “protection domains”
  – Liedtke suggests similar approach in the microkernel in an architecture-specific manner
    • PowerPC: use segment registers => no flush
    • Pentium or 486: share the linear hardware address space among several user address spaces => no flush
      – There are some caveats in terms of size of user space and how many can be “packed” in a 2**32 global space
• Conclusions
  – Address space switching among medium or small protection domains can ALWAYS be made efficient by careful construction of the microkernel
  – Large address spaces switches are going to be expensive ALWAYS due to cache effects and TLB effects, so switching cost is not the most critical issue
Thread switches and IPC

<table>
<thead>
<tr>
<th>System</th>
<th>CPU, MHz</th>
<th>RPC time (round trip)</th>
<th>cycles/IPC (oneway)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>full IPC semantics</td>
<td></td>
</tr>
<tr>
<td>L3</td>
<td>486, 50</td>
<td>10 µs</td>
<td>250</td>
</tr>
<tr>
<td>QNX</td>
<td>486, 33</td>
<td>76 µs</td>
<td>1254</td>
</tr>
<tr>
<td>Mach</td>
<td>R2000, 16.7</td>
<td>190 µs</td>
<td>1584</td>
</tr>
<tr>
<td>SRC RPC</td>
<td>CVAX, 12.5</td>
<td>464 µs</td>
<td>2900</td>
</tr>
<tr>
<td>Mach</td>
<td>486, 50</td>
<td>230 µs</td>
<td>5750</td>
</tr>
<tr>
<td>Amoeba</td>
<td>68020, 15</td>
<td>800 µs</td>
<td>6000</td>
</tr>
<tr>
<td>Spin</td>
<td>Alpha 21064, 133</td>
<td>102 µs</td>
<td>6783</td>
</tr>
<tr>
<td>Mach</td>
<td>Alpha 21064, 133</td>
<td>104 µs</td>
<td>6916</td>
</tr>
<tr>
<td></td>
<td></td>
<td>restricted IPC semantics</td>
<td></td>
</tr>
<tr>
<td>Exo-tlrpc</td>
<td>R2000, 16.7</td>
<td>6 µs</td>
<td>53</td>
</tr>
<tr>
<td>Spring</td>
<td>SparcV8, 40</td>
<td>11 µs</td>
<td>220</td>
</tr>
<tr>
<td>DP-Mach</td>
<td>486, 66</td>
<td>16 µs</td>
<td>528</td>
</tr>
<tr>
<td>LRPC</td>
<td>CVAX, 12.5</td>
<td>157 µs</td>
<td>981</td>
</tr>
</tbody>
</table>

Table 2: 1-byte-RPC performance
Segment switch (instead of AS switch) makes cross domain calls cheap
Memory Effects – System

Figure 3: Baseline MCPI for Ultrix and Mach.
Capacity induced MCPI

Figure 4: MCPI Caused by Cache Misses.
Portability vs. Performance

• Microkernel on top of abstract hardware while portable
  – Cannot exploit hardware features
  – Cannot take precautions to avoid performance problems specific to an arch
  – Incurs performance penalty due to abstract layer
Examples of non-portability

• Same processor family
  – Use address space switch implementation
    • TLB flush method preferable for 486
    • Segment register switch preferable for Pentium
      => 50% change of microkernel!
  – IPC implementation
    • Details of the cache layout (associativity) requires different handling of IPC buffers in 486 and Pentium

• Incompatible processors
  – Exokernel on R4000 (tagged TLB) Vs. 486 (untagged TLB)

=> Microkernels are inherently non-portable
Summary

• Minimal set of abstractions in microkernel
• Microkernels are processor specific (at least in implementation) and non-portable
• Right abstractions and processor-specific implementation leads to efficient processor-independent abstractions at higher layers