Microkernels and L4

Introduction

Why Microkernels?

Monolithic Kernel:
- Kernel has access to everything
  - all optimisations possible
  - all techniques/mechanisms/concepts implementable
- Can be extended by simply adding code
- Cost: Complexity
  - growing size
  - limited maintainability

Microkernel: Idea
- Small kernel providing core functionality
  - only code running in privileged mode
- Most OS services provided by user-level servers
- Applications communicate with servers via message-passing IPC

Trusted Computing Base
The part of the system which must be trusted to operate correctly

System:
- traditional
- embedded
- Linux/Windows

TCB:
- all code
- 100,000's loc

Microkernel-based
- 10,000's loc
**MICROKERNEL PROMISES**

- Combat kernel complexity, increase robustness, maintainability
  - dramatic reduction of amount of privileged code
  - modularisation with hardware-enforced interfaces
  - normal resource management applicable to system services
- Flexibility, adaptability, extensibility
  - policies defined at user level, easy to change
  - additional services provided by adding servers
- Hardware abstraction
  - hardware-dependent part of system is small, easy to optimise
- Security, safety
  - internal protection boundaries

**REALITY CHECK!**

- **First-generation microkernels**
  - Mach, Chorus, Amoeba
  - ... were slow...
  - 100µs IPC
  - almost independent of clock speed!
- **L4 does better**
  - close to hardware cost
  - 20 times faster than Mach on identical hardware

**IPC COST IMPLICATIONS**

**L4 IPC**

- 0.34 µs (Pentium 4 3GHz)
- 0.47 µs (Pentium 3 1.5GHz)
- 0.24 µs (Pentium 3 1.6GHz)
- 0.73 µs (Shannon 680HP Alpha)
- 0.34 µs (Shannon 680HP Alpha)
- 0.10 µs (UltraSPARC II/850MHz)
**MICROKERNEL PERFORMANCE**

**FIRST-GENERATION MICROKERNELS WERE SLOW**

- Reasons: Poor design [Liedtke SOSP 95]
  - complex API
  - too many features
  - poor design and implementation
  - large cache footprint ⇒ memory bandwidth limited

- L4 is fast due to small cache footprint
  - 10–14 I-cache lines
  - 8 D-cache lines
  - small cache footprint ⇒ CPU limited

**WHAT MAKES A MICROKERNEL FAST?**

- Small cache footprint, but how?
  - minimality: no unnecessary features
  - orthogonality: complementary features
  - well-designed, and well implemented from scratch!

- Kernel provides *mechanisms*, not *services*

- Design principle (minimality):

  A feature is only allowed in the kernel if this is required for the implementation of a secure system.

---

**L4 HISTORY**

- Original version by Jochen Liedtke (GMD) ≈ 93–95
  - “Version 2” API
  - i486 assembler
  - IPC 20 times faster than Mach [SOSP 93, 95]

- Other L4 V2 implementations:
  - L4/MIPS64: assembler + C (UNSW) 95–97
  - fastest kernel on single-issue CPU (100 cycles)
  - L4/Alpha: PAL + C (Dresden/UNSW), 95–97
    - first released SMP version
  - Fiasco (Pentium): C++ (Dresden), 97–99

---

**L4 HISTORY**

- Experimental “Version X” API
  - improved hardware abstraction
  - various experimental features (performance, security, generality)
  - portability experiments

- Implementations
  - Pentium: assembler, Liedtke (IBM), 97-98
  - Hazelnut (Pentium+ARM), C, Liedtke et al (Karlsruhe), 98–99
**L4 History**

- "Version 4" (X.2) API, 02
  - portability, API improvements
- L4Ka::Pistachio, C++ (plus assembler “fast path”)
  - x86, PPC-32, Itanium (Karlsruhe), 02–03
  - fastest ever kernel (36 cycles, NICTA/UNSW)
  - MIPS64, Alpha (NICTA/UNSW) 03
  - same performance as V2 kernel (100 cycles single issue)
  - ARM, PPC-64 (NICTA/UNSW), x86-64 (Karlsruhe), 03-04
  - UltraSPARC (NICTA/UNSW), 04–??
- Portable kernel:
  - ≈ 3 person months for core functionality
  - 6–12 person months for full functionality & optimisation

**L4 Present**

- NICTA L4-embedded commercially deployed
  - adopted by Qualcomm for CDMA chipsets
  - under evaluation/development for other products at a number of multinationals
  - about to establish strong presence in wireless and CE markets
- NICTA spinning out Open Kernel Labs
  - further development of L4-embedded
  - professional services for L4 users
  - commercialisation of present NICTA microkernel research

**L4 Future**

- Security API: NICTA seL4
  - draft published March 06
  - semi-formal specification in Haskell
  - "executable spec": Haskell implementation plus ISA simulator
  - used for exercising and porting apps
  - stable API August 06
  - C implementation end of 06
  - similar project at TU Dresden: L4sec (draft API Oct 05)
- Features:
  - user-level management of kernel resources (esp. memory)
  - low-overhead information-flow control mechanisms
  - suitable for formal verification
- Formal verification of L4 implementation: L4.verified project
  - mathematical proof that implementation matches spec
**PISTACHIO: SIZE**

- Source code:
  - \(\approx 10k\) loc architecture independent
  - \(\approx 0.5\)–\(2k\) loc architecture specific
- Memory footprint kernel (no attempt to minimise yet):
  - using gcc (poor code density on RISC/EPIC architectures)
- Fast IPC cache footprint (typical):
  - 10–14 I-cache lines
  - 8 D-cache lines

**Size Comparison**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Version</th>
<th>Text</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>L4Ka</td>
<td>52k</td>
<td>98k</td>
</tr>
<tr>
<td>Itanium</td>
<td>L4Ka</td>
<td>173k</td>
<td>417k</td>
</tr>
<tr>
<td>ARM</td>
<td>NICTA</td>
<td>55k</td>
<td>117k</td>
</tr>
<tr>
<td>PPC-32</td>
<td>L4Ka</td>
<td>41k</td>
<td>135k</td>
</tr>
<tr>
<td>PPC-64</td>
<td>L4Ka</td>
<td>60k</td>
<td>205k</td>
</tr>
<tr>
<td>MIPS-64</td>
<td>L4Ka</td>
<td>61k</td>
<td>100k</td>
</tr>
<tr>
<td>ARM</td>
<td>NICTA</td>
<td>55k</td>
<td>117k</td>
</tr>
<tr>
<td>Pentium-3</td>
<td>UKa</td>
<td>180</td>
<td>367</td>
</tr>
<tr>
<td>Small Spaces</td>
<td>UKa</td>
<td>385</td>
<td>983</td>
</tr>
<tr>
<td>Pentium-4</td>
<td>UKa</td>
<td>508</td>
<td>508</td>
</tr>
<tr>
<td>Itanium 2</td>
<td>UKa/NICTA</td>
<td>7419</td>
<td>7410</td>
</tr>
<tr>
<td>cross CPU</td>
<td>UKa</td>
<td>276</td>
<td>276</td>
</tr>
<tr>
<td>MIPS64</td>
<td>NICTA/UNSW</td>
<td>3238</td>
<td>3238</td>
</tr>
<tr>
<td>PowerPC-64</td>
<td>NICTA/UNSW</td>
<td>330</td>
<td>518</td>
</tr>
<tr>
<td>Alpha 21264</td>
<td>NICTA/UNSW</td>
<td>440</td>
<td>642</td>
</tr>
<tr>
<td>ARM/XScale</td>
<td>NICTA/UNSW</td>
<td>340</td>
<td>340</td>
</tr>
</tbody>
</table>

**PISTACHIO PERFORMANCE: IPC**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>port/optimisation</th>
<th>C++ optimised</th>
<th>optimised</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>intra AS</td>
<td>inter AS</td>
<td>intra AS</td>
</tr>
<tr>
<td>Pentium-3</td>
<td>UKa</td>
<td>180</td>
<td>367</td>
</tr>
<tr>
<td>Small Spaces</td>
<td>UKa</td>
<td>385</td>
<td>983</td>
</tr>
<tr>
<td>Pentium-4</td>
<td>UKa/NICTA</td>
<td>508</td>
<td>508</td>
</tr>
<tr>
<td>Itanium 2</td>
<td>UKa</td>
<td>7419</td>
<td>7410</td>
</tr>
<tr>
<td>cross CPU</td>
<td>UKa</td>
<td>276</td>
<td>276</td>
</tr>
<tr>
<td>MIPS64</td>
<td>NICTA/UNSW</td>
<td>3238</td>
<td>3238</td>
</tr>
<tr>
<td>PowerPC-64</td>
<td>NICTA/UNSW</td>
<td>330</td>
<td>518</td>
</tr>
<tr>
<td>Alpha 21264</td>
<td>NICTA/UNSW</td>
<td>440</td>
<td>642</td>
</tr>
<tr>
<td>ARM/XScale</td>
<td>NICTA/UNSW</td>
<td>340</td>
<td>340</td>
</tr>
</tbody>
</table>

\(^1\) "Version 2" assembler kernel

**L4 ABSTRACTIONS AND MECHANISMS**

**Three basic abstractions:**
- Address spaces
- Threads
- Time (second-class abstraction in N2 API, to vanish completely)

**Two basic mechanisms:**
- Inter-process communication (IPC)
- Mapping
L4 Abstractions: Address Spaces

- Address space is unit of protection
  - Initially empty
  - Populated by mapping in frames
- Mapping performed by privileged MapControl() syscall
  - Can only be called from root task
  - Also used for revoking mappings (unmap operation)
- Root task
  - Initial address space created at boot time
  - Controls system resources
  - Non-delegatable privilege (shortcoming of N2 API)

L4 Abstractions: Threads

- Thread is unit of execution
  - Kernel-scheduled
- Thread is addressable unit for IPC
  - Thread-ID is unique identifier
- Threads managed by user-level servers
  - Creation, destruction, association with address space
- Thread attributes:
  - Scheduling parameters (time slice, priority)
  - Unique ID
  - Address space
  - Page-fault and exception handler

L4 Abstractions: Time

- Used for scheduling time slices
  - Thread has fixed-length time slice for preemption
  - Time slices allocated from (finite or infinite) time quantum
    - Notification when exceeded
- In earlier L4 versions also used for IPC timeouts
  - Removed in N2

L4 Mechanism: IPC

- Synchronous message-passing operation
  - Data copied directly from sender to receiver
    - Short messages passed in registers
- Can be blocking or polling (fail if partner not ready)
- Asynchronous notification variant
  - No data transfer, only sets notification bit in receiver
    - Receiver can wait (block) or poll
**L4 Concepts: Root Task**

- First task started at boot time
- Can perform *privileged system calls*
- Controls access to resources
  - threads
  - address spaces
  - physical memory

**L4 Exception Handling**

- **Interrupts**
  - modelled as hardware “thread” sending messages
  - received by registered (user-level) interrupt-handler thread
  - interrupt acknowledged when handler blocks on receive
  - timer interrupt handled in-kernel
- **Page faults**
  - kernel fakes IPC message from faulting thread to its pager
  - pager requests root task to set up a mapping
  - pager replies to faulting client, message intercepted by kernel
- **Other exceptions**
  - kernel fakes IPC message from exceptor thread to its exception handler
  - exception handler may reply with message specifying new IP, SP
  - can be signal handler, emulation code, stub for IPCing to server, ...

**Features Not In Kernel**

- System services (file system, network stack, ...)
  - implemented by user-level servers
- VM management
  - performed by (hierarchy) of user-level pagers
- Device drivers
  - user-level threads registered for interrupt IPC
  - map device registers