
Lecture 3: OS Structure II

CSC 469H1F

Fall 2006

Angela Demke Brown



Recap: Microkernels

- Design philosophy
 - Small privileged kernel provides core function
 - Most OS services provided by user-level servers
- Promise
 - Less complex kernel → more robust, maintainable
 - Dramatically less privileged code
 - Hw-enforced interfaces between modules
 - Flexibility, customizability, extensibility
- Reality: What went wrong?

IPC Costs

- First generation microkernels were slow
 - Mach, Chorus, Amoeba
 - 100 microsecs IPC (almost independent of CPU clock speed!)
 - Many concluded this was inherent limitation of microkernel approach
- Second generation microkernels tackled IPC performance head on
 - L4 (Jochen Liedtke @ Karlsruhe, Gernot Heiser @ UNSW)
 - 20 times faster than Mach on same hardware

Example of IPC Performance

- "Improving IPC by Kernel Design" by J. Liedtke, Proceedings of the 14th SOSP, December 1993.

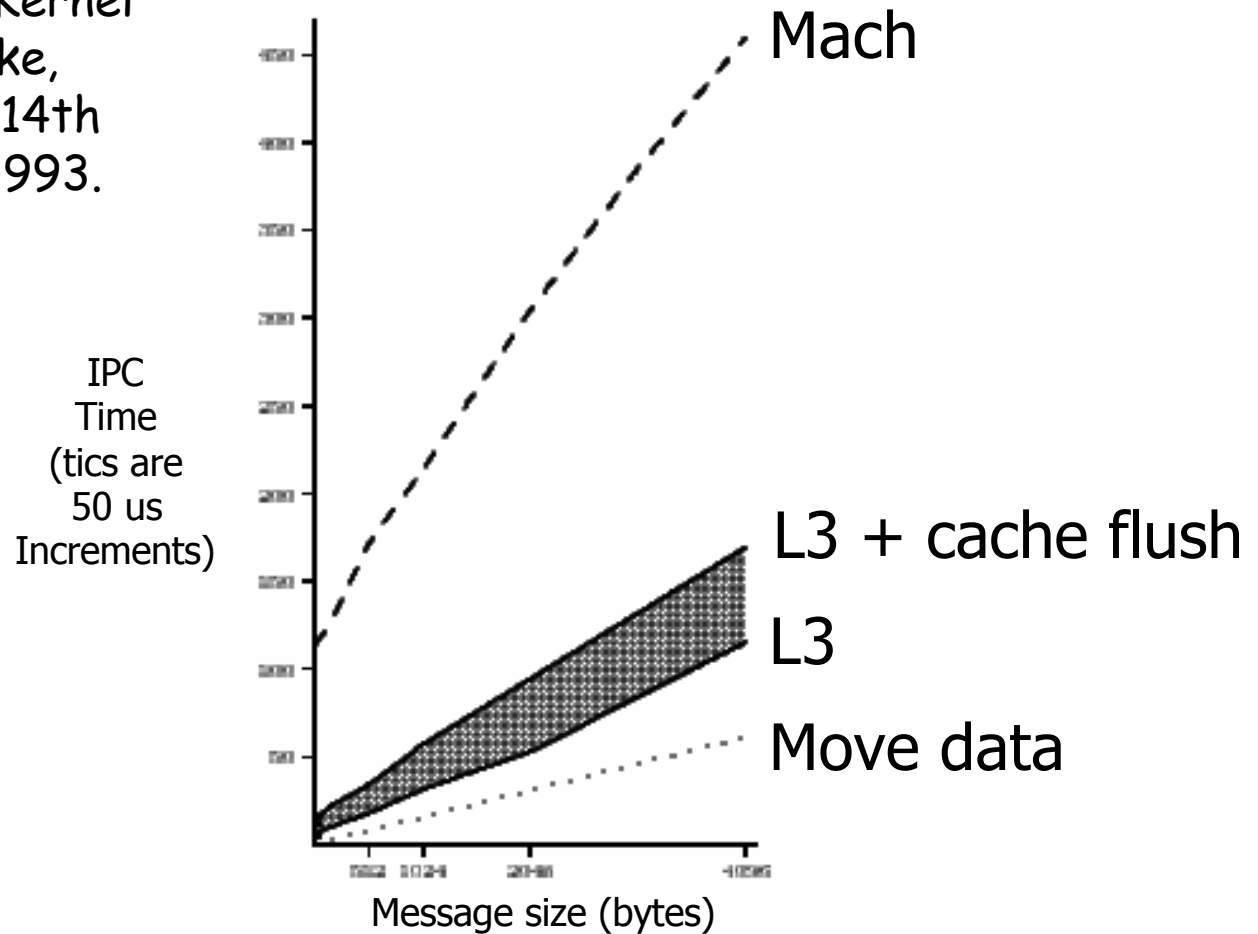


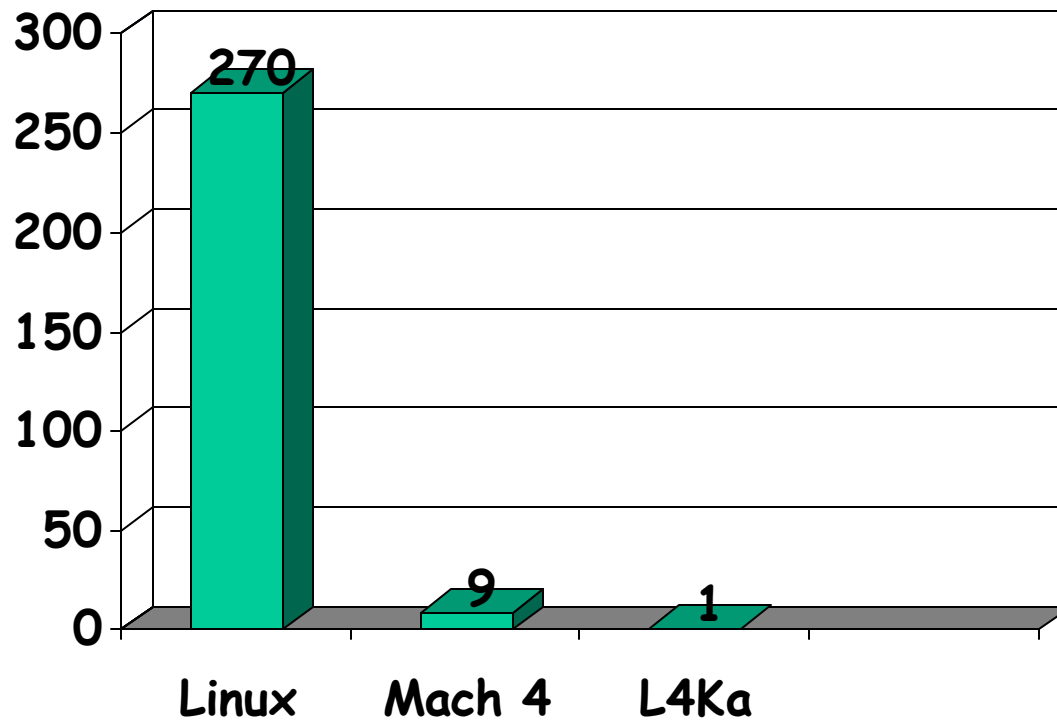
Figure 8: 486-DX50, L3 versus Mach Ipc Times

Why the difference?

- First generation poorly designed (Liedtke)
 - Complex API
 - Too many features
 - Large cache footprint → memory bw limited
- L4 is fast due to small cache footprint
 - 10-14 I-cache lines
 - 8 D-cache lines
 - Small cache footprint → CPU limited
 - L4 + user-level Linux server 5-7% slower than native Linux

Size Comparison

- Lines of code (x 10,000)



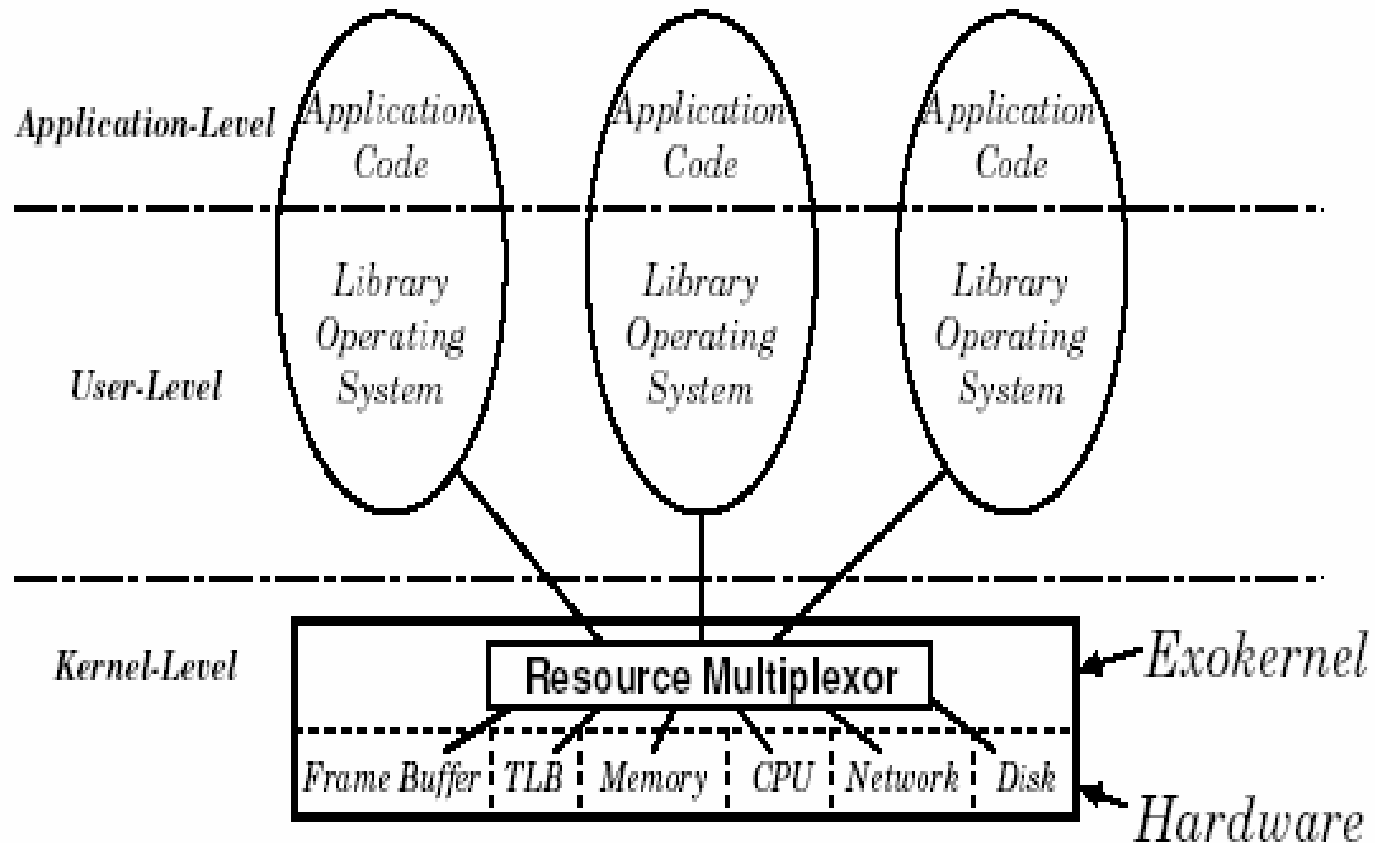
L4 Abstractions & Mechanisms

- Two basic abstractions (in latest version)
 - Address spaces - unit of protection
 - initially empty
 - Populated by privileged mapping operating
 - Threads - unit of execution
 - Kernel-scheduled, user-level managed
- Two basic mechanisms
 - IPC - synchronous message passing
 - Mapping - all access to memory, devices

How far can we take this?

- Microkernels: minimal set of abstractions and mechanisms
- Exokernel: MIT Research project
 - OS abstractions are bad
 - Deny application-specific optimizations
 - Discourage innovation
 - Impose "mandatory costs"
 - Separate concept of protection from abstraction
- Exokernel is a resource multiplexor

Exokernel Architecture



Exokernel basics

- Interface is low-level (expose HW, kernel data structures)
- Fine-grained resource multiplexing (i.e., individual disk blocks, not disk partitions)
- Management is limited to protection
- Revocation of resources is visible to user-level libOS
- Untrusted Deterministic Functions (UDFs) allow exokernel to check properties of metadata

Going farther...

- Exokernel drops OS abstractions, multiplexes hardware
- Much like an older strategy... Virtual Machines
 - Place thin layer of software "above" hardware
 - virtual machine monitor (VMM, hypervisor)
 - Exports raw hardware interface
 - OS/application above sees "virtual" machine identical to underlying physical machine
 - VMM multiplexes virtual machines

VM Examples

- Original - IBM's VM/CMS (1970's)
- Now hot again:
 - Disco (stanford research) → VMWare
 - Denali (U. of Washington)
 - Xen (Cambridge)
- What's the big deal about virtual machines?