

Microkernels: Mach and L4

Presented by Jason Wu

With content borrowed from Dan Williams
(2009) and Hakim Weatherspoon (2008)

Outline

- Introduction to Kernels
- 1st Generation Microkernels
 - Mach
- 2nd Generation Microkernels
 - L4
- Conclusions

Introduction to Kernels

- Different Types of Kernel Designs
 - Monolithic kernel
 - Microkernel
 - Hybrid Kernel
 - Exokernel
 - Virtual Machines?

Monolithic Kernels

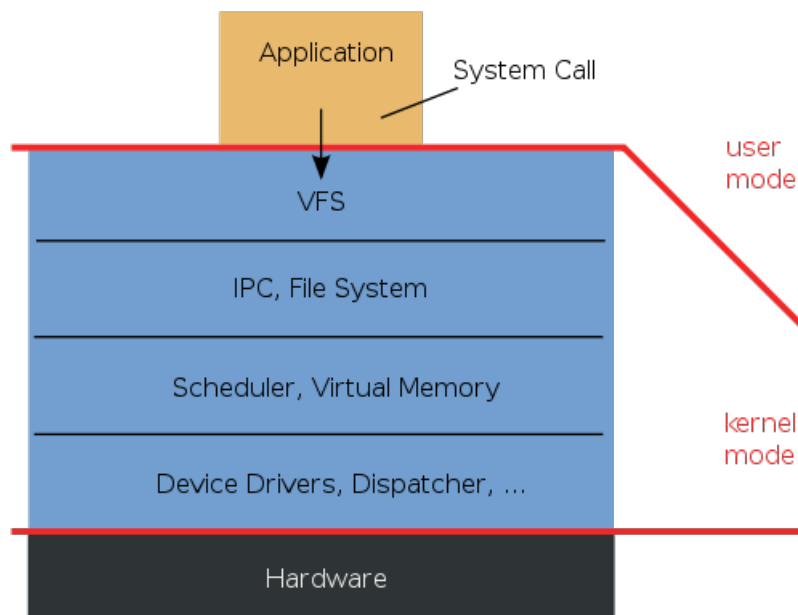
- All OS services operate in kernel space
- Good performance
- Disadvantages
 - Dependencies between system component
 - Complex & huge (millions(!) of lines of code)
 - Larger size makes it hard to maintain
- E.g. Multics, Unix, BSD, Linux

Microkernels

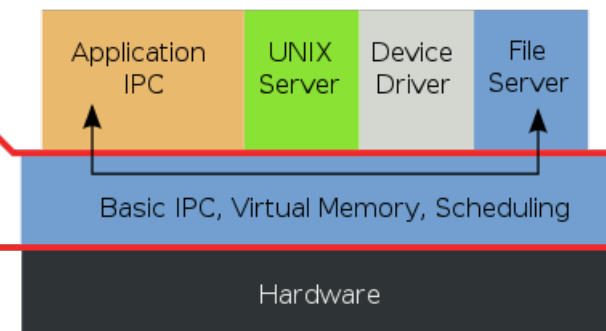
- Minimalist approach
 - IPC, virtual memory, thread scheduling
- Put the rest into user space
 - Device drivers, networking, file system, user interface
- More stable with less services in kernel space
- Disadvantages
 - Lots of system calls and context switches
- E.g. Mach, L4, AmigaOS, Minix, K42

Monolithic Kernels VS Microkernels

**Monolithic Kernel
based Operating System**



**Microkernel
based Operating System**



Hybrid Kernels

- Combine the best of both worlds
 - Speed and simple design of a monolithic kernel
 - Modularity and stability of a microkernel
- Still similar to a monolithic kernel
 - Disadvantages still apply here
- E.g. Windows NT, NetWare, BeOS

Exokernels

- Follows end-to-end principle
 - Extremely minimal
 - Fewest hardware abstractions as possible
 - Just allocates physical resources to apps
- Disadvantages
 - More work for application developers
- E.g. Nemesis, ExOS
- Next Thursday!

The Microkernel Debate

- How big should it be?
- Big debate during the 1980's

Summary: Kernels

- Monolithic kernels
 - Advantages: performance
 - Disadvantages: difficult to debug and maintain
- **Microkernels**
 - Advantages: more reliable and secure
 - Disadvantages: more overhead
- Hybrid Kernels
 - Advantages: benefits of monolithic and microkernels
 - Disadvantages: same as monolithic kernels
- Exokernels
 - Advantages: minimal and simple
 - Disadvantages: more work for application developers

1st Generation Microkernels

The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System

- SOSP 1987
- Young et al

- Richard Rashid
 - Lead developer of Mach
 - Microsoft Research

- William Bolosky
 - Microsoft Research

Mach

- 1st generation microkernel
- Based on Accent
- Memory object
 - Manage system services like network paging and file system
- Memory via communication

Mach Abstractions

- Task
 - Basic unit of resource allocation
 - Virtual address space, communication capabilities
- Thread
 - Basic unit of computation
- Port
 - Communication channel for IPC
- Message
 - May contain port capabilities, pointers
- Memory Object

External Memory Management

- No kernel-based file system
 - Kernel is just a cache manager
- Memory object
 - AKA “paging object”
- Page
 - Task that implements memory object

Lots of Flexibility

- E.g. consistent network shared memory
 - Each client maps X with shared pager
 - Use primitives to tell kernel cache what to do
 - Locking
 - Flushing

Problems of External Memory Management

- External data manager failure looks like communication failure
 - E.g. need timeouts
- Opportunities for data manager to deadlock on itself

Performance

- Does not prohibit caching
- Reduce number of copies of data occupying memory
 - Copy-to-use, copy-to-kernel
 - More memory for caching
- “compiling a small program cached in memory... is twice as fast”
- I/O operations reduced by a factor of 10
- Context switch overhead?

2nd Generation Microkernels

The Performance of Micro-Kernel- Based Systems

- SOSP 1997
- Hartig et al
- Jochen Liedtke
 - Worked on Eumel, L3

The Performance of Micro-Kernel- Based Systems

- Evaluates the L4 microkernel
- Ports Linux to run on top of L4
- Suggests improvements

L4

- 2nd generation microkernel
- Similar to Mach
 - Started from scratch, rather than monolithic
 - Even more minimal
- Uses user-level pages
- Tasks, threads, IPC

L4Linux

- Linux source has two cleanly separated parts
 - Architecture dependent
 - Architecture independent
- In L4Linux
 - Architecture dependent code is modified for L4
 - Architecture independent part is unchanged
 - L4 not specifically modified to support Linux

L4Linux

- Linux kernel as L4 user service
 - Runs as an L4 thread in a single L4 address space
 - Creates L4 threads for its user processes
 - Maps parts of its address space to user process threads (using L4 primitives)
 - Acts as pager thread for its user threads
 - Has its own logical page table
 - Multiplexes its own single thread (to avoid having to change Linux source code)

L4Linux – System Calls

- The statically linked and shared C libraries are modified
 - Systems calls in the lib call the Linux kernel using IPC
- For unmodified native Linux applications, there is a “trampoline”
 - The application traps
 - Control bounces to a user-level exception handler
 - The handler calls the modified shared library
 - Binary compatible

A Note on TLBs

- A Translation Look-aside Buffer (TLB) caches page table lookups
- On context switch, TLB needs to be flushed
- A tagged TLB tags each entry with an address space label, avoiding flushes
- A Pentium CPU can emulate a tagged TLB for small address spaces

Performance - Benchmarks

- Compared the following systems
 - Native Linux
 - L4Linux
 - MkLinux (in-kernel)
 - Linux ported to run inside the Mach microkernel
 - MkLinux (user)
 - Linux ported to run as a user process on top of the Mach microkernel

Performance - Microbenchmarks

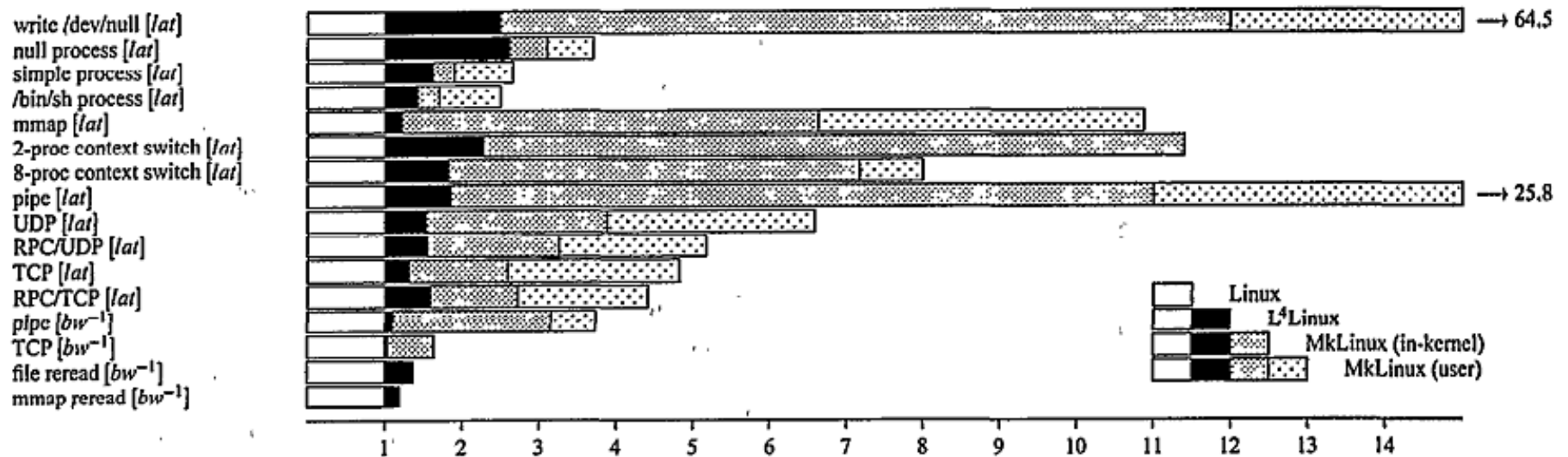


Figure 6: *Imbench* results, normalized to native Linux. These are presented as slowdowns: a shorter bar is a better result. [lat] is a latency measurement, [bw⁻¹] the inverse of a bandwidth one. Hardware is a 133 MHz Pentium.

Performance - Macrobenchmarks

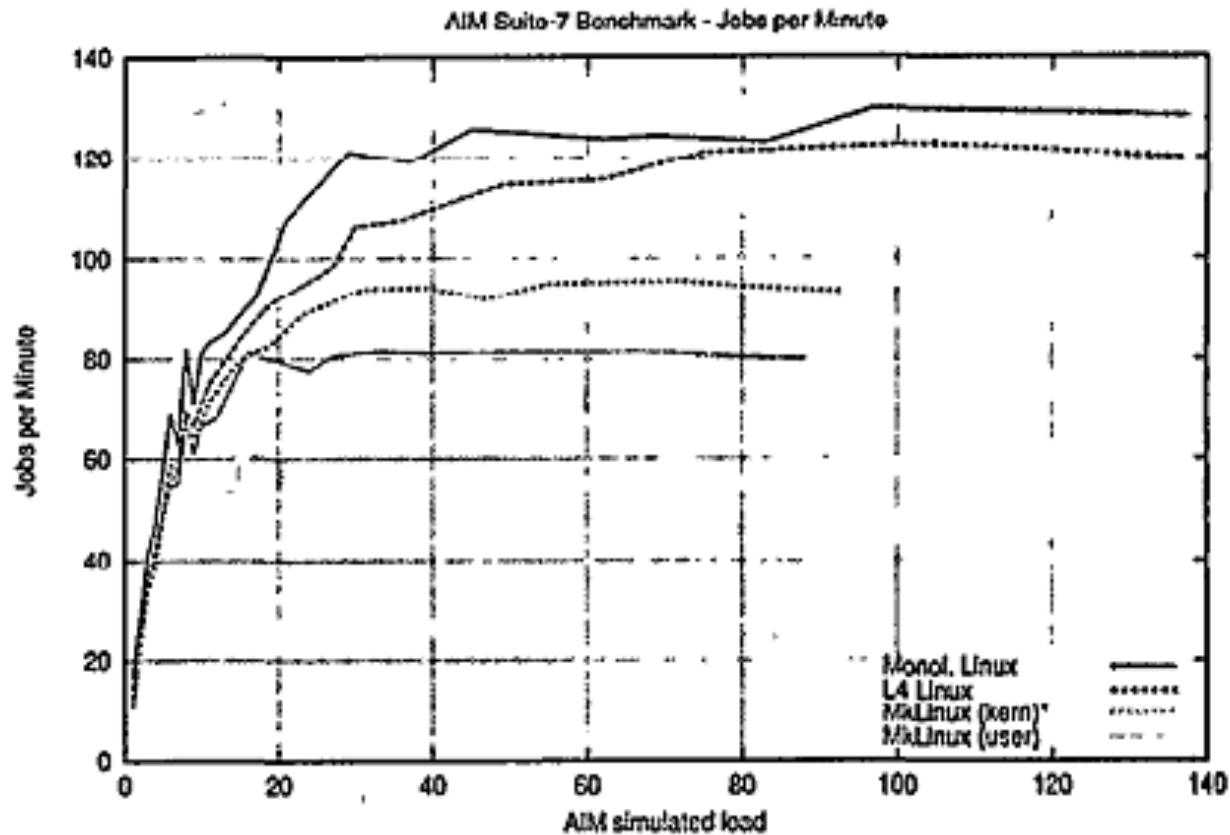


Figure 9: *AIM Multiuser Benchmark Suite VII*. Jobs completed per minute depending on AIM load units. (133 MHz Pentium)

Performance - Analysis

- L4Linux is 5% - 10% slower than native Linux for macrobenchmarks
- User mode MkLinux is 49% slower (averaged over all loads)
- In-kernel MkLinux is 29% slower (averaged over all loads)
- Co-location of kernel is not enough for good performance

L4 is Proof of Concept

- Pipes can be made faster using L4 primitives
- Linux kernel was essentially unmodified
 - Could be optimized for microkernel
- More options for extensibility

Conclusions

- Microkernels have attractive properties
 - Extensibility benefits
 - Minimal/simple
- Microkernels have comparable performance