Microkernels: Mach and L4

CS 6410: Advanced Systems

Fall 2025

Hakim Weatherspoon

Introduction to Kernels

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

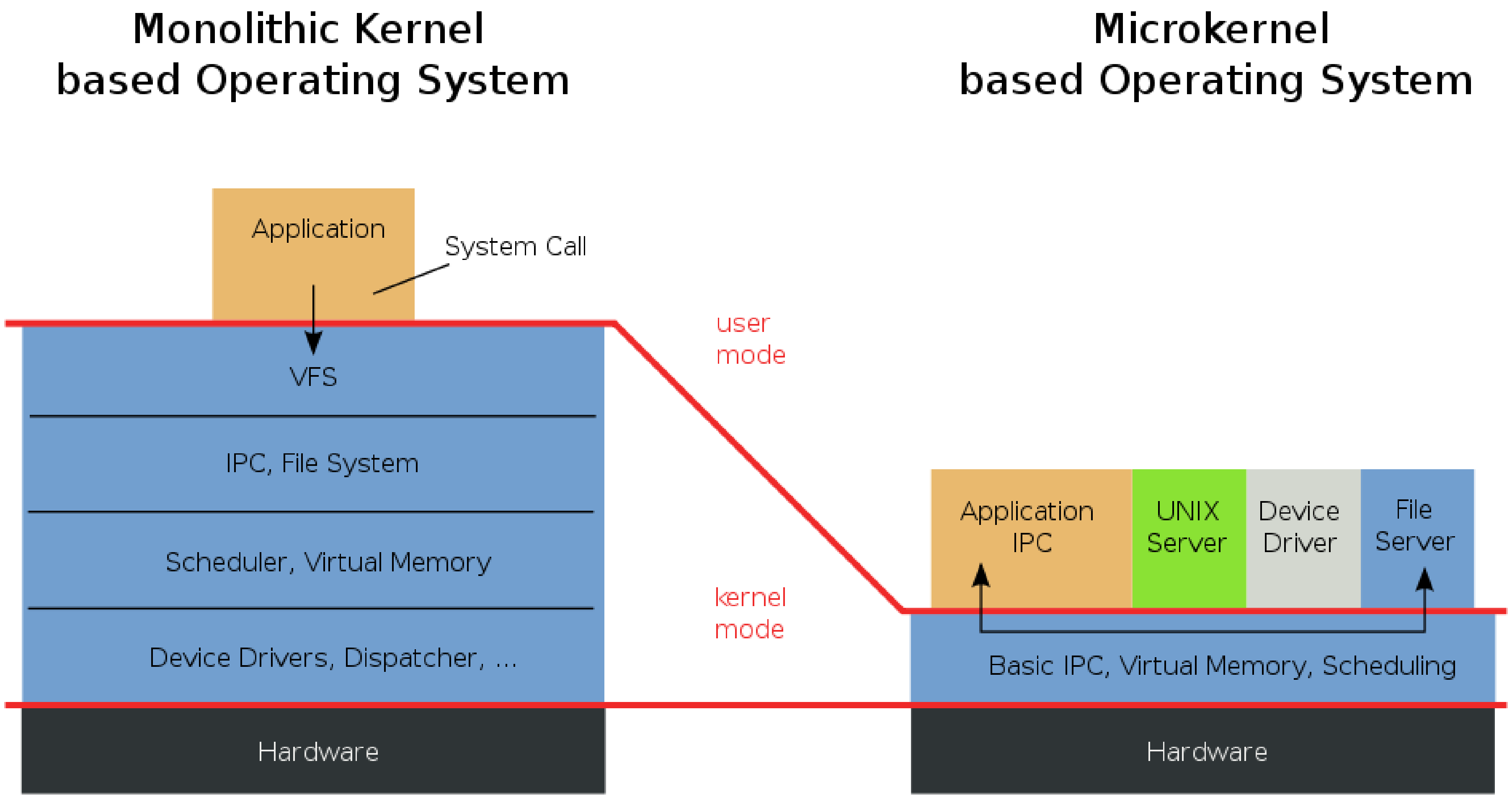
Monolithic Kernels

- 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, even the pager for virtual memory
- 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



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
- This Thursday!

The Microkernel Debate

- How big should it be?

- Big debate during the 1980's
  - Michael Stonebraker publishes Operating System Support for Database Management
    - File system, scheduler, concurrency control all suboptimal for databases
    - Perhaps different applications require different OS primitives

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

Mach: A New Kernel Foundation For UNIX Development

- USENIX Summer Conference 1986
- Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young

- Richard Rashid
  - Lead developer of Mach
  - Microsoft Research

- William Bolosky
  - Microsoft Research
- Avadis Tevanian
  - Primary figure in development of Mac OS X
  - Apple Computer (former VP and CTO)

Mach

- 1^{st} generation microkernel
- Based on Accent
- Memory object
    - Mange 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"
- Pager
  - 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
- Herman Hartig, Michael Hohmuth, Jochen Liedtke, Sebastian Schonberg, Jean Wolter

- Herman Hartig
  - Prof at TU Dresden

- Jochen Liedtke
  - Worked on microkernels Eumel, L3
  - Is the "L" in L3 and L4

Mach was terrible…

- Mach (and other gen-1 microkernels) performed so terribly that no one took the idea seriously
- High overheads prevented adoption as a substrate for monolithic kernels
- High overheads would negate any benefits from creative new systems built directly on Mach

…but L4 could be better

- IPC needs to be 1-2 orders of magnitude faster
- Address space switches need to be less costly

L4 design goals

- Hardware Abstraction
- Modularity
- Security
- Performance
  - L4 really wanted to prove that microkernels were better than monolithic kernels
- Servicing page faults and syscalls over the network, again?

IPC

- Really was an order of magnitude faster than Mach in the first generation (and it pretty much only got faster)
- Which enabled using IPC to handle hardware interrupts and syscalls

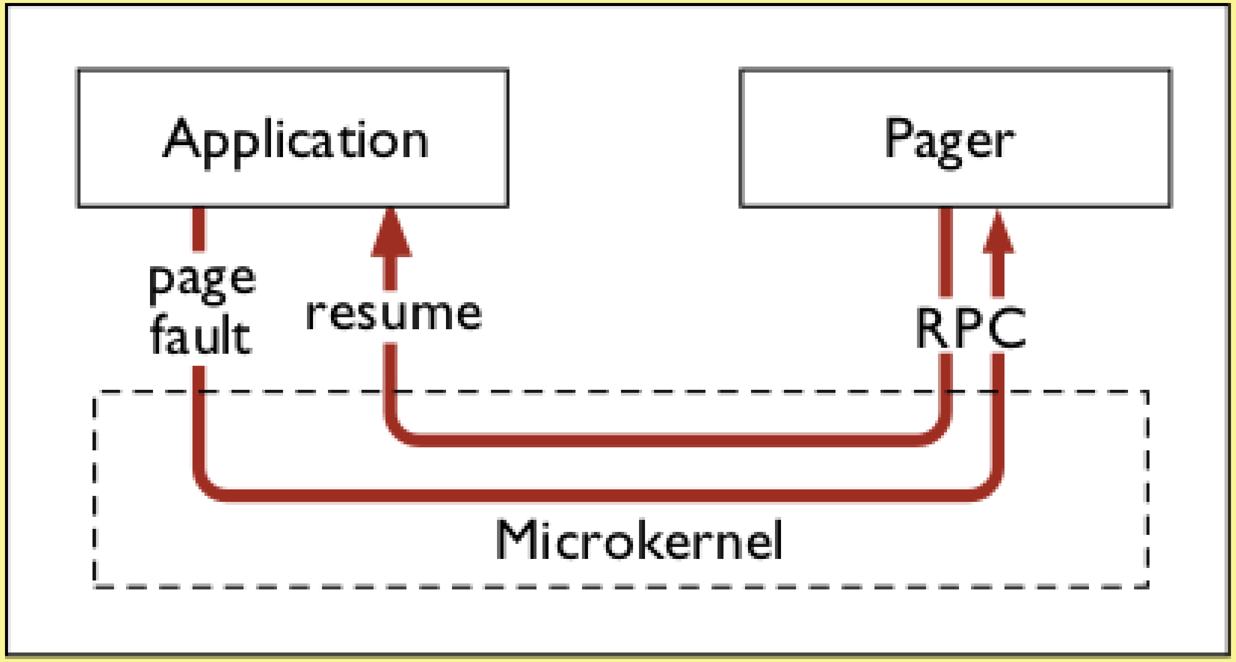


Figure 1. Page fault processing

Address Spaces

- L4 allowed recursive address space construction
- Any process could grant a page in its own address space to another consenting process
  - (this removed the page from the granter's address space)
- Any process could map a page in its own address space to another consenting process
  - (this kept the page in the granter's address space, creating a shared page)
- Processes could unmap any page in their own address space from all process who had inherited the page directly or indirectly from the unmapper
- In this way user space servers could perform almost all memory management functions

Address Spaces

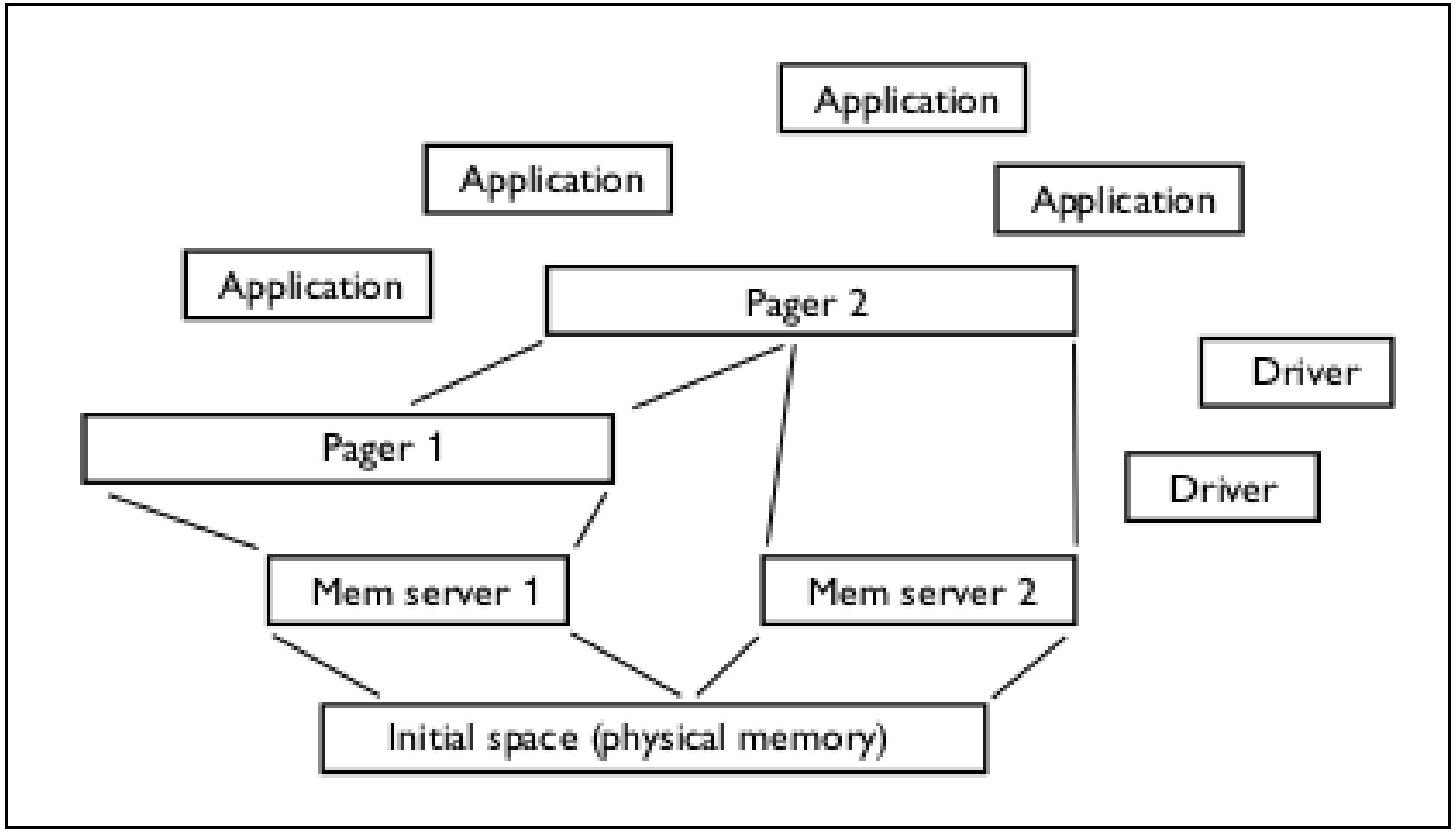


Figure 6. A maps page by IPC to B

Address Spaces

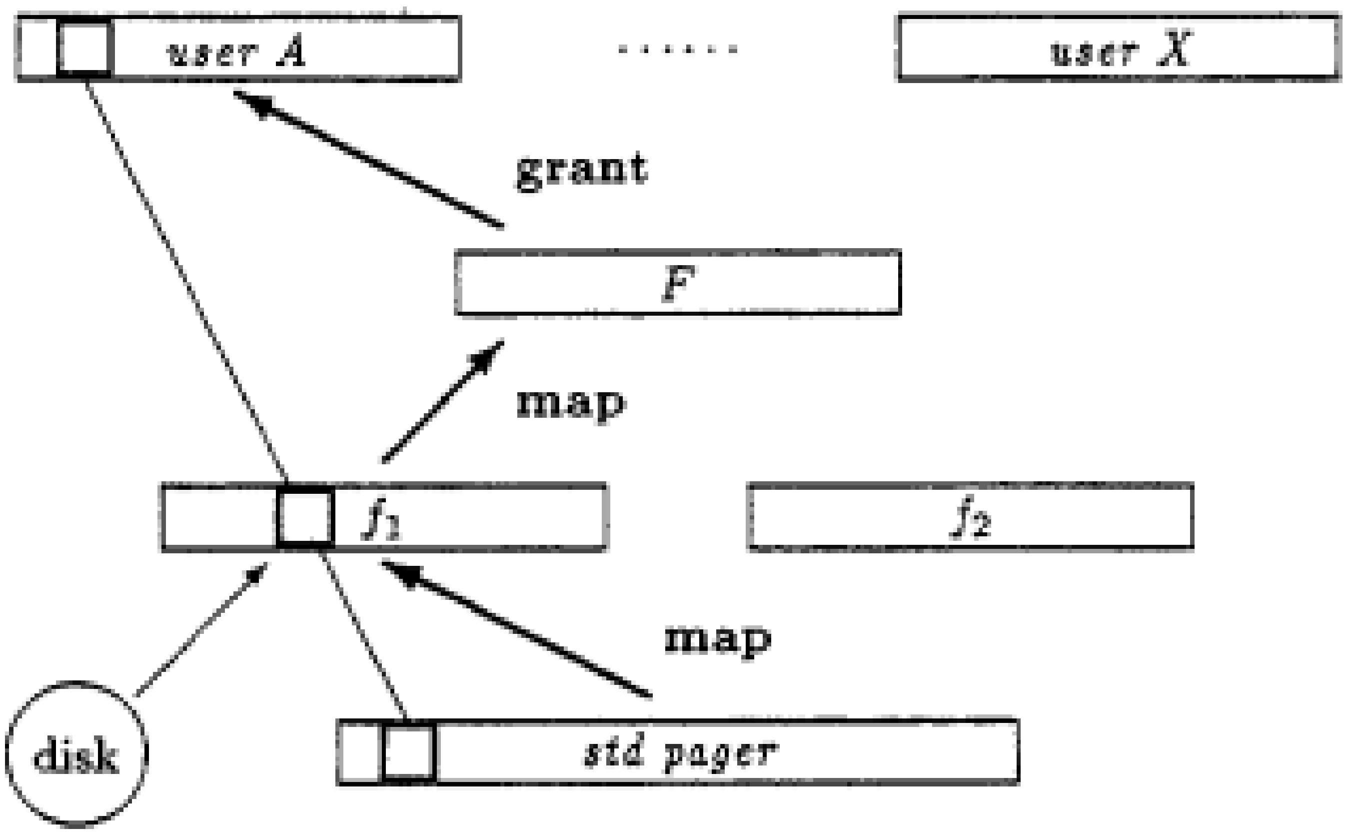


Figure 1: *A Granting Example.*

Discussion Questions

- Do these abstractions seem flexible enough to support any operating system on top of them performantly?
- Flexible enough to implement any system someone might care to actually build?

The Performance of Micro-Kernel-Based Systems

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

L4

- 2^{nd} 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

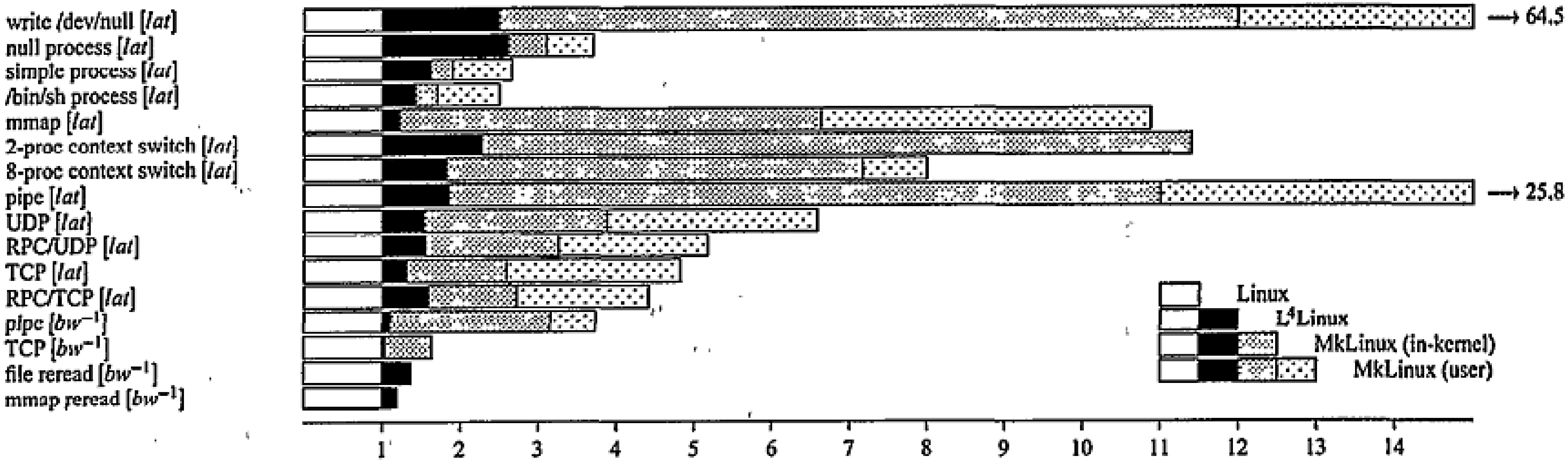


Figure 6: *lmbench results, normalized to native Linux.* These are presented as slowdowns: a shorter bar is a better result. [*lat*] is a latency measurement, [bw^{-1}] the inverse of a bandwidth one. Hardware is a 133 MHz Pentium.

Performance - Macrobenchmarks

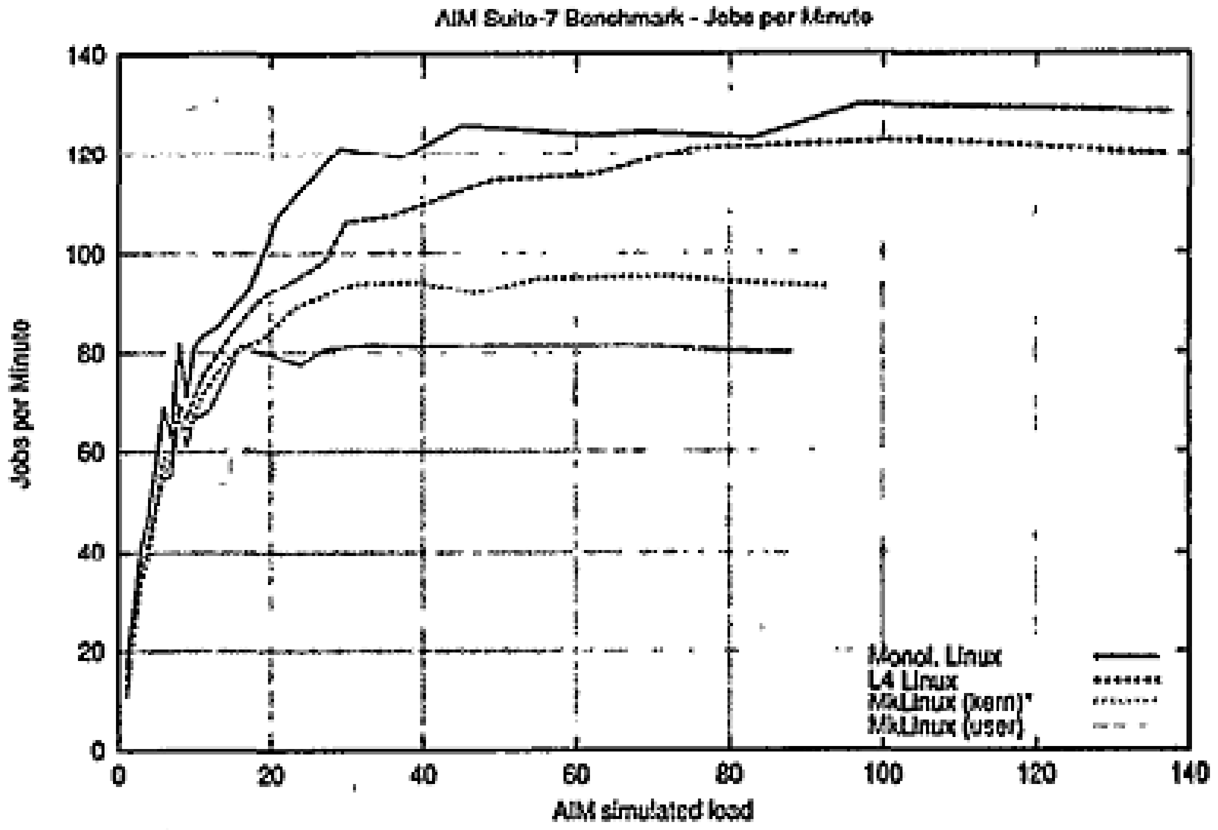


Figure 9: *AIM Multiuser Benchmark Suite VII.* Jobs completed per minute depending on AIM load units. (133 MHz Pentium)

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

Perspective

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

Next Time

- Project: next step is the Survey Paper

- Read and write a review:
  - Exokernel: an operating system architecture for application-level resource management, Dawson R. Engler, M. Frans Kaashoek, and James O'Toole, Jr. In *ACM symposium on Operating systems principles (SOSP)*, December 1995, pages 251–266
    https://dl.acm.org/doi/10.1145/224057.224076

  - *Optional:* Extensibility, Safety and Performance in the SPIN Operating System , Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers, and Susan Eggers. In *ACM symposium on Operating systems principles (SOSP)*, December 1995, pages 267 – 283
    https://dl.acm.org/doi/abs/10.1145/224056.224077