Exokernel
An Operating System Architecture for Application-Level Resource Management

Josh Triplett
March 2, 2006

Purpose of OS design
  • All about applications
  • Users don’t want to run an OS
  • Users want to run their applications
  • Kernels from an application perspective

What does an application need?
  • Protection from other applications
  • Services
  • Access to hardware resources
  • How are those services provided?
  • How fast are those services?
  • How well do those services match application needs?

Monolithic kernel
  • Kernel has one address space
  • No boundaries within kernel
  • All services built into kernel
  • System calls relatively expensive
  • Kernel internal operation fast
• One-size-fits-all services
• Abstractions impair efficiency
• Especially if not the right abstractions

Microkernel

• "minimal" kernel
• Simpler, lower-level abstractions over hardware
• Processes and timesharing
• Address spaces, paging and virtual memory
• Inter-process communication (IPC)
• Additional services provided via IPC
• Good protection, isolation
• Customizable

Microkernel problems

• Extensive IPC impacts performance
• More generally, abstractions still impair efficiency
• "Minimal abstraction" is an oxymoron
• Still providing some one-size-fits-all services
• Concept good: move services out of kernel
• Could we go further?

Exokernel

• Secure hardware multiplexer
• Expose details of hardware resources
• Validate access
• Everything else provided by "Library operating systems"
Expose resource allocation and naming

- Library OS requests specific physical resources
- Physical pages, disk blocks, time slices, etc
- Expose physical resource names/numbers
- No implicit allocation
- Mechanism, not policy
- Library OSes can implement various policies

Expose resource revocation

- Exokernel needs a unit of a resource
- Example: memory page
- Exokernel asks a library OS
- “Free a page”
- Library OS chooses what to free
- Exokernel sets time limits
- Time’s up: forcibly revoke, notify

Hardware to multiplex

- CPU
- Interrupts
- Memory
- Direct Memory Access (DMA)
- Disk
- Network
- Control transfer
CPU

- Resource: linear vector of CPU time
- Mechanism: safely expose timer interrupts
- Partitions time into time slices
- Library OS reserves specific time slices in advance
- Various scheduling policies
- Long, infrequent slices for throughput
- Short, frequent slices for responsiveness
- Context saving and switching implemented by library OS
- Like scheduler activations, but more efficient

Physical memory

- Resource: Linear physical memory
- Mechanism: Safely expose TLB and/or page table
- Library OSes request pages
- Exokernel validates access
- Library OS controls protection, sharing
- Library OS handles caching, locality, etc
- Exokernel provides safe DMA

Network

- Resource: incoming data stream
- Mechanism: packet filter
- Filters compiled into machine code by exokernel
- Filters run safely in kernel
Protected control transfer

- Client transfers control to server
- Predefined server entry point
- Client donates time slice
- No other functionality
- Highly optimized: 30 instructions
- Library OSes implement context saving if desired
- Can use to build IPC, RPC, pipes
- Or not: just a control transfer

Library operating systems

- Build on top of exokernel hardware interfaces
- Provide functionality to applications
- Functionality and policies specifically tuned to application needs
- May provide full OS services
- May provide minimal services, maximal performance
- Application interface to library OS can be efficient
- Exokernel invocation efficient: 18 instructions
- Good platform for experimentation

Exokernel summary

- Problem: Abstractions are slow, not tuned to all applications
- Solution: Just provide hardware multiplexing, nothing else
- Expose physical resource allocation, naming, revocation
- Support library operating systems
- Mechanism, not policy
- Highly efficient
- Highly flexible