Exokernel
An Operating System Architecture for Application-Level Resource Management
Operating Systems Purpose

Users don’t want to run an OS
They want to run applications
We should design OS kernels with application needs in mind
What Do Applications Need?

Protection from other applications
Services
Access to hardware that may require privilege
How Good Is The OS?

How are services provided?
How fast are the services?
How well do they match application needs?
What’s Wrong With Monolithic OS?

All services built into the kernel
No internal protection boundaries
Internal operation is fast, but system calls are quite expensive
One size fits all services
Abstractions may not be suitable, and they impair efficiency
What’s Wrong With Microkernels?

Minimal kernel is good
Lower-level abstractions are an improvement
Extensibility is good
Potential for protection and fault isolation
Boundary crossing via IPC is expensive
Minimal abstraction is an oxymoron
Exokernel

Secure hardware multiplexor
Expose hardware, don’t abstract it
Don’t virtualize it either!
Kernel validates access
The rest of the OS is in a library
Resource Allocation

Library OS requests specific resources
Page frames, disk blocks, time slices ...
Exokernel exposes resource names
No hidden allocation (no virtualization)
All mechanism, no policy
Visible Revocation

Exokernel asks for resources to be returned
i.e. free a page
Library OS chooses what to free
Revocation is forced if library OS is not responsive
Multiplexed Hardware

CPU
Interrupts
Memory
DMA
Disk
Network
CPU

Resource: linear time vector
Mechanism: expose timer interrupts
Granularity: time slices
Library OS reserves future time slices
Context switching implemented by library OS
Similar to scheduler activations
Physical Memory

Resource: linear physical memory
Mechanism: safely expose TLB & page tables
Library OS requests pages
Exokernel validates access
Network

Resource: incoming data stream
Mechanism: packet filter
Filters compiled into machine code by exokernel
Filters safely colocated with the Exokernel
Protected Control Transfer

Client transfers control to predefined server entry point
Client donates scheduling time slice
That’s it! ... only 30 instructions
Exokernel provides the minimal mechanism
Could build RPC, IPC, on top of this ... or not
Library Operating Systems

Reside in application address space
Use exokernel interfaces
Exokernel invocation takes 18 instructions
Invocation of library OS from applications is a procedure call
OS specifically tuned for application needs
Conclusion

Don’t abstract or virtualize, it’s inefficient! Just provide secure hardware multiplexing. Expose physical resources. Expose names, allocation and revocation. OS should provide minimal mechanism, not policy.