Today’s Paper

Traditional Operating Systems

- Traditional operating systems fix the interface and implementation of OS abstractions

---

Example: Exokernel

- Apache
  - Library OS Chosen from available
  - Abstractions
  - Interface

- SQL Server
  - Library OS Customized for SQL Server
  - Abstractions
  - Interface

- Exokernel
  - Hardware
Problems with Traditional OS

- **Performance**
  - Denies applications the advantages of domain-specific optimizations
- **Flexibility**
  - Restricts the flexibility of application builders
  - Concept: "with more information exposed, resources can be utilized 'better'!"
- **Functionality**
  - Discourages changes to the implementations of existing abstractions

Solution: Exokernel

- **Separate protection from management!**
  - Allow user level to manage resources
    - Application libraries implement OS abstractions
  - Exokernel exports resources
    - Low level interface
    - Protects, does not manage
    - Expose hardware

- End-to-end argument; “applications know better”
Exokernel + Library OS

- **Exokernel**'s resource management:
  - Allocate, revoke, share, track ownership

- **Library OS**:
  - Uses low-level Exokernel interface, provides higher-level abstractions; provides special purpose implementations

An application can choose the library which best suits its needs, or even build its own.
Exokernel

• Hypotheses:
  – Exokernels can be very efficient
  – Low-level, secure multiplexing of hardware resources can be implemented efficiently
  – Traditional operating system abstractions can be implemented efficiently at application level
  – Applications can create special-purpose implementations of these abstractions

Library Operating Systems

• Simpler
• Specialized
• Multiple can exist
• Few kernel crossings
**Design Challenge**

- How can an Exokernel allow libOSes to freely manage physical resources while protecting them from each other?
  - Track ownership of resources
    - Secure bindings – libOS can securely bind to machine resources
  - Guard all resource usage
    - Invisible/visible resource revocation
  - Revoke access to resources
    - Abort protocol

**Design Principles**

- Securely expose hardware
  - Kernel should provide secure low-level primitives that allow all hardware resources to be accessed as directly as possible.
- Expose allocation
  - Allow to request specific physical resources
- Expose names
  - Export physical names.
  - Remove a level of indirection: Translation
- Expose revocation
  - Utilize a visible resource revocation protocol
Secure Bindings

- Exokernel allows LibOSes to bind resources using secure bindings
- Decouples authorization from the actual use of a resource
- Multiplex resources securely
- Performs authorization only at bind time
  - Allows the kernel to protect resources without having to understand them

Some Terminology

- Packet filters
- TLB
- Downloadable code (ASH)
- RPC
- DMA
Kernel Comparisons

Microkernels

• A good idea in the 1970s and 80s
• Not up to demands of modern processors
  • Virtual memory
  • Heavy caching
• Not up to demand of modern operating systems
• Resurrection:
  – Mobile phones, PDAs, handheld devices
    – Fixed or limited functionality
    – No general purpose files
    – No dynamic virtual memory

⇒
  – Simple context switches
  – All code already in memory
  – Easy IPC