

- What's the objective of an operating system?
- How is it achieved?

Exokernel

- Paper describes:
 - exokernel idea
 - Aegis exokernel
 - ExOS library OS

Consider this...

- Process has 11 pages memory map, Processor has 10 pages memory available for this process, OS manages physical memory with *a* page replacement policy... Which one?

or

- Process is about to lock a shared data structure, but the OS is going to preempt it immediately afterwards. What could we do?
- A trend which we'll see in other papers too:
 - Application knows best!

Possible (bad) solutions

- Permit applications to do whatever they want
- Check always with applications what they want OS to do
- ...

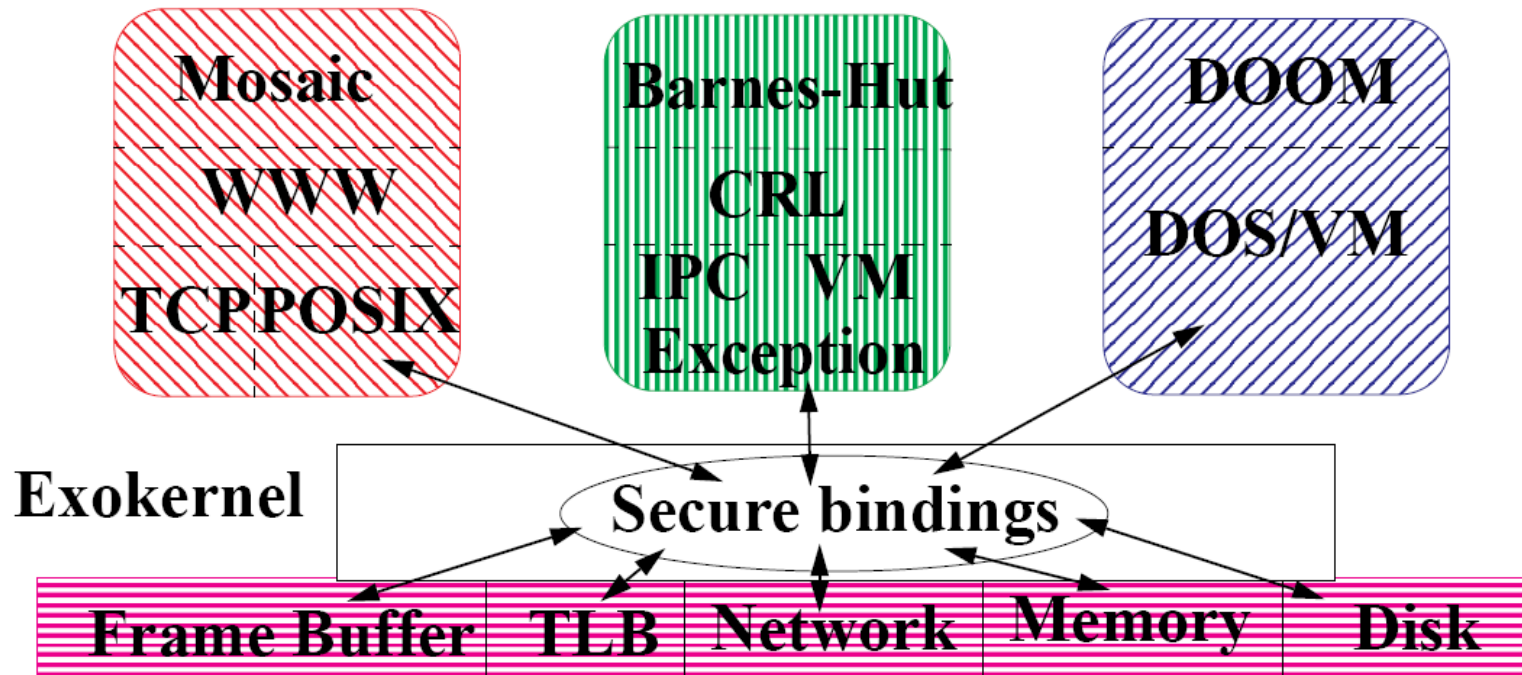
Other (better) solutions

- Define right set of abstractions so applications can build arbitrary services
 - Process or thread, address space or page, file or block...
 - What's the right set of abstractions?
- Let applications change what OS does
 - By changing/specializing/extending the OS kernel
 - By modularizing OS and replacing modules
 - What ends up in core module?
- Let applications make decisions, just verify that they had the rights/permission to make that decision
 - Resource protection vs. resource management
 - The basic idea of exokernel approach: separate the two

Exokernel – key ideas

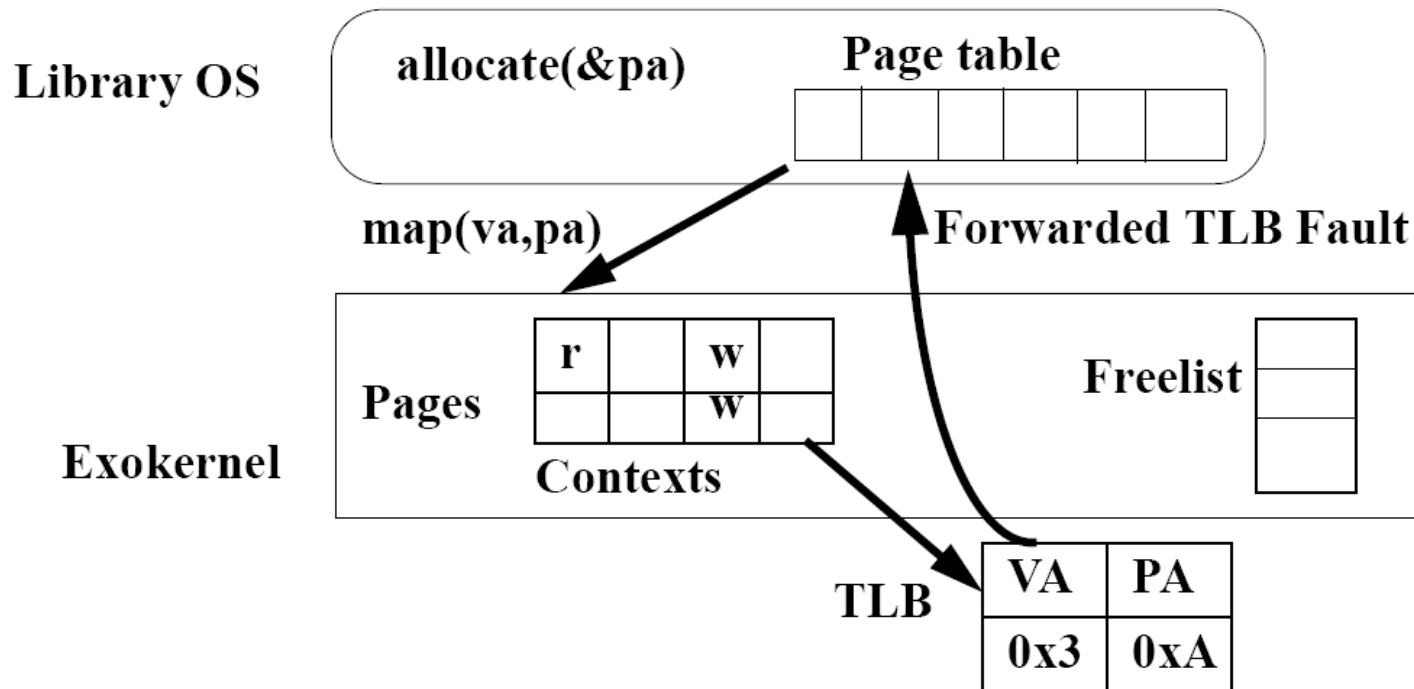
- Separation of resource protection from resource management
- Low-level interface – expose hardware
 - Fine-grained hardware multiplexing
 - CPU slice, TLB entry or TLB, hardware page table entry, memory page, also interrupts and exceptions...
 - Applications pay for what they need
 - offers flexibility and performance
 - Interface == pseudo instructions related to access/manipulation of protected resources
- Let applications form secure binding to resource, then track all resource usage and binding points
- Support visible revocation and abort protocol
- Expose kernel data structures, names...
 - (read only)
- Support safe kernel extensions

Exokernel – high level view



- Applications and OS libraries
- Exokernel – thin layer responsible for safe multiplexing
- Physical resources

Virtual Memory example



- When binding established authentication, etc. take place
- When binding is used, page accessed, no added overheads

Secure binding

- Guard everything and track ownership of everything
- Policy and mechanisms for determining access in libOS/application
 - Must have some initial bootstrapping allocation however
- Provide capability – secure binding
 - Leverage hardware support whenever possible, otherwise software
- Perform fast access checks

Multiplexing resources

- Utilize hardware, software, both
- Let applications specify exception/interrupt handlers
 - Processor context per application to include exception and interrupt handling info (+ other stuff...)
 - => visible revocation
- CPU
 - Maintain time vector; use timer interrupts to tell application of imminent context switch so “right” resources can be saved
- Memory
 - Rely on hardware address translation; add software TLB to expand set of fast VA->PA checks
- Network
 - Trickier; to parse a packet, need higher-level info to parse it;
 - Allow dynamic packet filters (DPFs) to be dynamically generated and inserted into kernel; checked for safety and sandboxed
 - ASHs – Application-specific Handlers – addition additional functionality in kernel
- Forceful revocation – abort protocol

More details...

- Control transfer
 - Exceptions
 - Protected control transfers between address spaces
 - Save by default minimum amount of state
 - Leverage shared memory for fast inter process communication
- Like exceptions, exokernel maintains “protected entry context” with info on entry points into an address space
- Also address translation context for select guaranteed mappings
- Aegis in physical memory

Aegis Processor Environment

- Exception Context
- Interrupt Context
- Protected Entry Context
 - for synchronous and asynchronous protected control transfers
- Addressing Context:
 - set of guaranteed mappings (including for all of the above), address space id, status register and a hash into Aegis software TLB

Examples

- UNIX low-level IPC: pipes, shm; RPC built on top
- ExOS: pipes, shm, and lightweight RPC built on top of protected control transfer supported by exokernel
 - performance benefit order of magnitude+
 - why?
- Support for trusted RPC
- Scheduling algorithms
- Benefits of ASHs