EXOKERNELS

Dawson Engler, Frans Kaashoek, James O'Toole

MIT Laboratory for Computer Science

http://www.pdos.lcs.mit.edu/

Ambitious applications must fight OS

- **OS abstractions preempt application design decisions since:**
  - **No perfect implementation exists**
  - **No perfect interface exists**
  - **You cannot avoid them**
- **Result: applications run slowly or can’t be written**

Exokernel: maximize application freedom

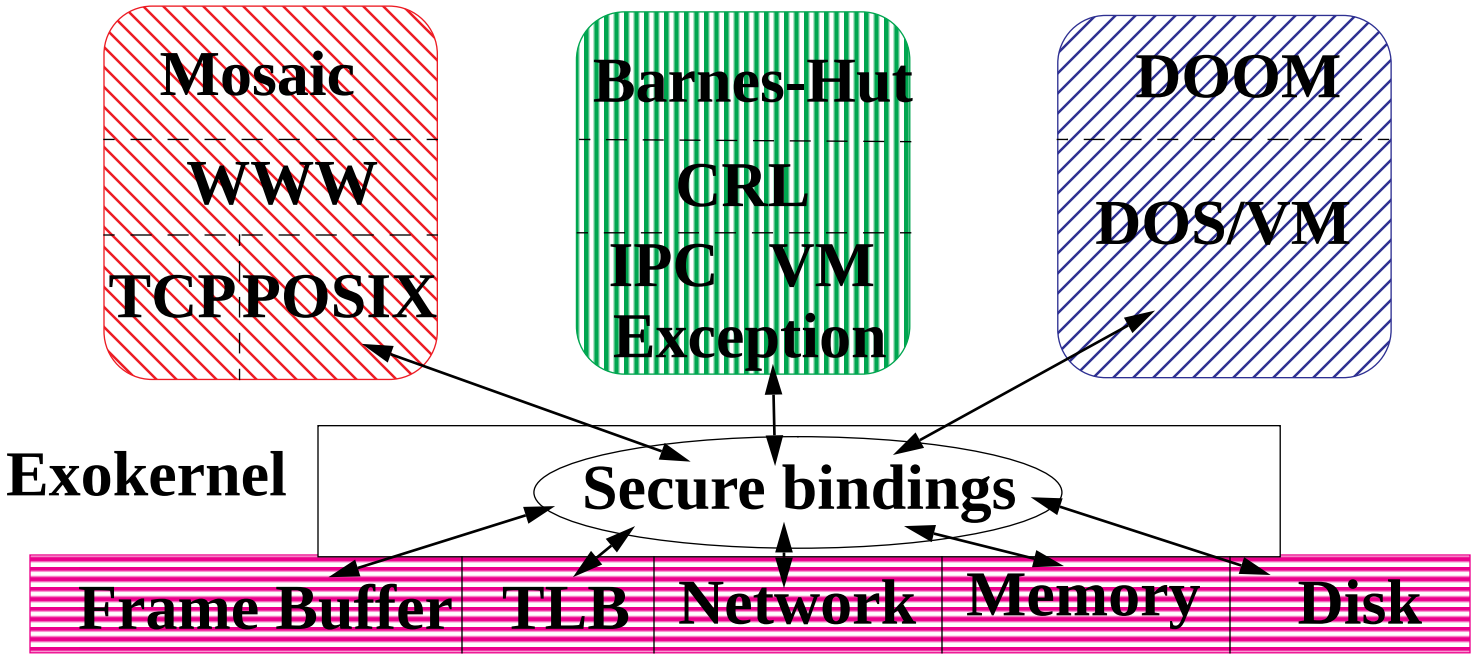
- Insight: implement OS abstractions at application-level
- How: securely multiplex hardware without abstracting it

  Export hardware to applications

  Protection by thin OS veneer, the exokernel

  System objects and policies in untrusted libraries
- Result: Can do operations impossible on traditional systems

Exokernel architecture



- An exokernel safely exports hardware to applications
- Applications build abstractions with library OSs
- Shared libraries to reduce space consumption

Advantages of library operating systems

- **Tightly coupled to applications**
  - **Simple specialized implementations**
  - **New abstractions**
  - **Libraries can trust applications**
- **Fast system evolution**
  - **Library operating systems can be developed in isolation**
  - **Anyone can develop, modify library OS**
  - **Crucial: # of application writers >> kernel hackers**

Challenges of application control

- **Preventing system chaos**

  Use standards and good programming methodology

- **Portability**

  Standard solution: hardware abstraction layer

- **Decentralizing services**

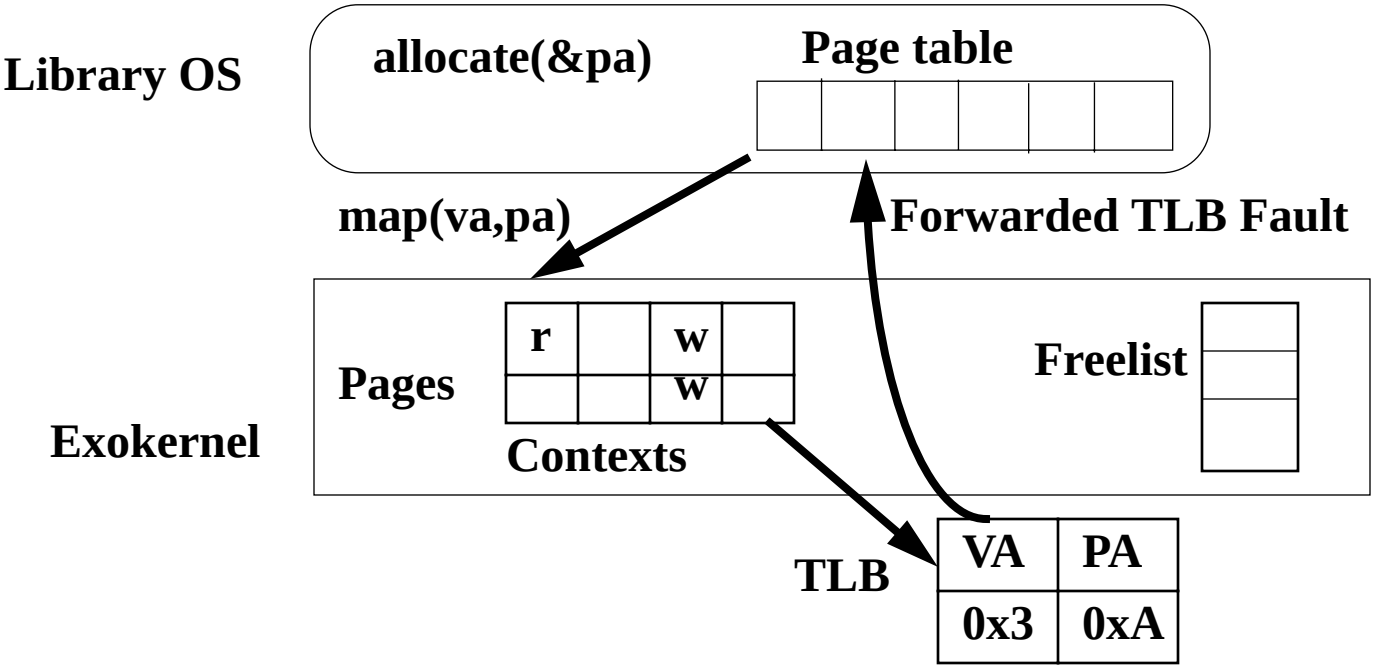
  Use kernel protection mechanism to replace servers

- **Reconcile global performance with local optimization**

Maximize application control

- Fine-grain multiplexing of hardware
- Low-level interface
- Limit kernel resource management to protection
- Revoke resources visibly
- Expose kernel data structures, hardware, and names

Example: virtual memory

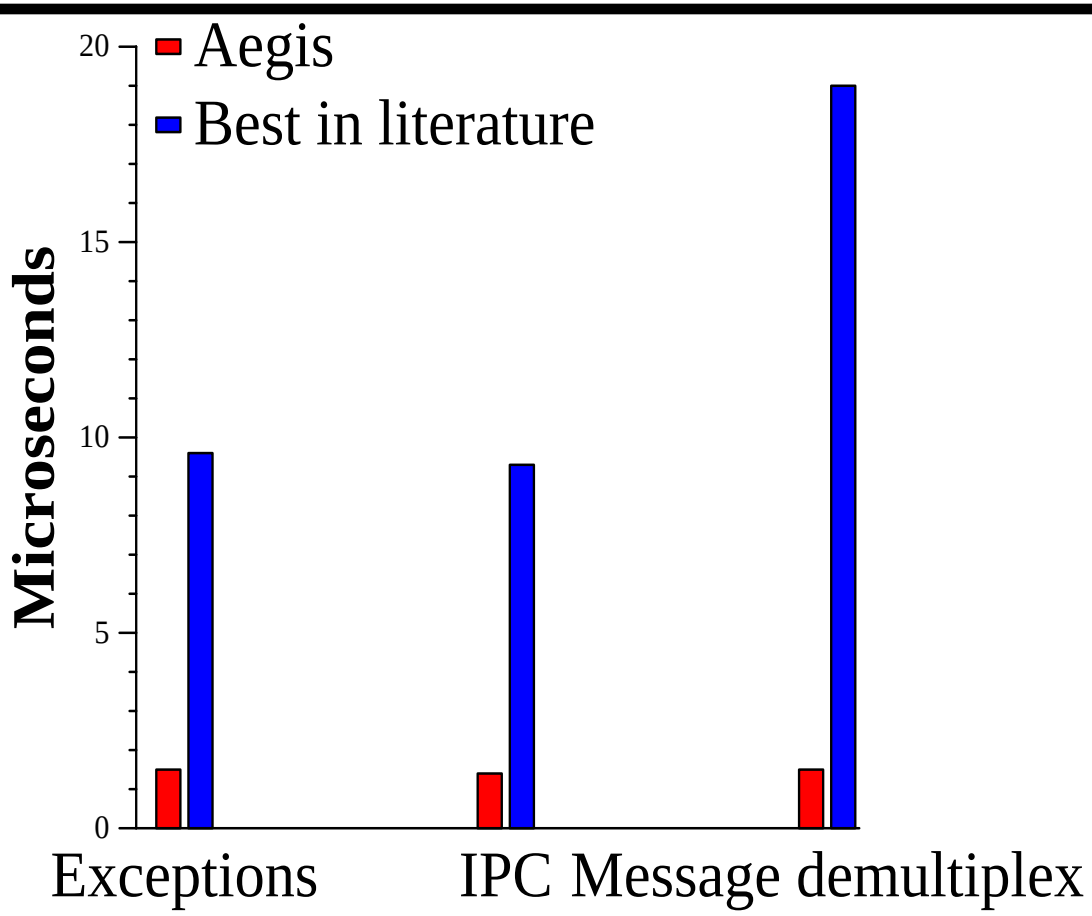


- Revocation: ask application for physical page
- If application is uncooperative, take pages by force

Aegis: A prototype MIPS exokernel

- Physical memory
- TLB entries
- Time slices
- Network
- Environments (process as defined by hardware):
  1. Interrupt/exception forwarding vector
  2. Pinned virtual mappings
  3. Control transfer entry points

Aegis performance on DEC5000/25Mhz



- **Easy to make simple operations fast**

ExOS: an application-library OS

- Rudimentary UNIX-like library
- Completely implemented in application space
  1. Processes (fork, exec)
  2. IPC (signals, LRPC, shared memory, and pipes)
  3. Virtual memory (sbrk, mmap, shared memory)
  4. Exception handling (fast user-level traps)
  5. Networking (IP protocols, NFS, Sun RPC)

Application-level VM (AVM)

- **Flexibility examples with AVM:**
  1. **Fine-grained monitoring (hardware is visible)**
  2. **Tighter integration with other abstractions**

- **Performance (times in μsec):**

OS	Dirty	Appel1	Appel2
Ultrix	n.a.	262	232
ExOS	9.8	34	22

Extending AVM

- Linear versus clustered page-tables [SOSP 95]
- Performance (times in μsec):

Structure	(un)Prot1	Prot100	Appel1	Appel2
Cluster	13	238	25	15
Linear	16.9	109	34	22

- Easy to modify library OS:

  Implemented in library in few weeks by undergrad

  Difficult to do even as trusted user on traditional OS

Extensibility example: Trusted LRPC

- Trusted LRPC: trust server to save and restore registers

LRPC system	Performance (μsec)
Traditional LRPC	6.9
Trusted LRPC	2.9

- Pay for what you need

Summary of paper results

- **Simple kernel primitives are fast**
- **OS abstractions can be efficiently implemented as libraries**
- **Extensibility gives substantial performance benefits**

Summary

- **Problem: OS abstractions hurt applications**
- **Insight: Securely export hardware without abstracting it**
- **How: Exokernel architecture**
  - **Kernel interface at hardware level**
  - **OS abstractions in untrusted libraries**
- **Advantages:**
  - **Can do operations not possible on traditional systems**
  - **Large implementor base = fast system evolution**