Instructor Notes

Exokernel: An Operating System Architecture for Application-level Resource Management
Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.
In the Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95), Copper Mountain Resort, Colorado, December 1995, pages 251-266.

1. What is the overriding goal of exokernel? How is this different than what previous extensible systems did (Nucleus, Mach, HYDRA)? How is this different from a VVM?

- Separate protection from management
- lowest level primitives possible:
  more efficient
  more flexibility
  - expose hardware resources

- Similar goals, but more extreme

Compare - don't try to abstract resources

or virtualize them (like VMM)

- not even "mechanism"

2. What are exokernel's design principles?

- securely expose hw
  - instr.
  - resources
  (avoid resource management)
  let libOS do
- expose allocation
  let apps request physical resources
  (don't make implicit)
  - this cpu?
  - this page?
- expose names
- expose revocation
  - if something taken away, let app know

3. Where does policy belong? How does exokernel handle conflicts/competing applications?

- in libOS
- if managed by same libOS → fine
- otherwise ??
  - "must contain policy to arbitrate btwn competing libOSes"

4. What is the purpose of a **secure binding?** Why can secure bindings achieve good performance? How can secure bindings be used to multiplex physical memory? To multiplex the network?

- guard resources efficiently
- decouples authorization from use

1) Primitives for setting up should be good match w/ hw (a exokernel)
2) only need to be ~~done~~ setup (and understand semantics) @ bind, not access

- Mem: check capability to access page on TLB miss (fast access)
  ↑ binding
- Network:

  multiplex network requires interpreting incoming msgs

  → download "safe" code into kernel to do this interpretation

5. What is an Application-Specific Safe Handler (ASH) and why is it useful?

- Code run in kernel on behalf of app
- Faster w/o switches out

6. Why is resource revocation **visible** to applications using an exokernel?

- Traditional OS: invisible - just take time slice or page away
- Exo: believe app should be able to respond (save regs on context switch) (page out w/ phys. mem)

7. Why is an **abort protocol** needed?

- If libOS doesn't revoke as desired

1) Could kill libOS

2) Tell libOS gone
   + Guarantee some minimal resources

8. How did the authors demonstrate the flexibility of the exokernel architecture?

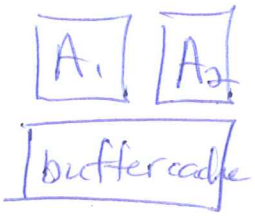
1) Aegis is efficient w/ base costs

Section 7) ExOS

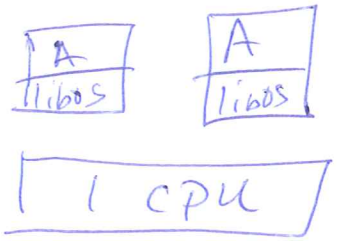
- extensible RPC (who saves regs)
- inverted page table (instead of linear)
- extensible scheduler
  - stride

9. How can various resources be "multiplexed" without policy decisions?

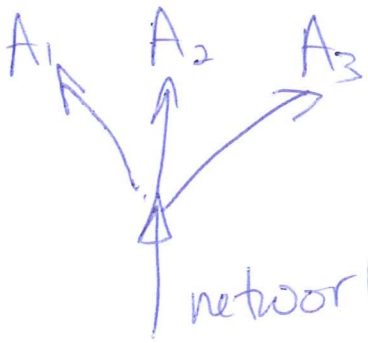
point of mux is decision point!

2) 

should they get to share?

1) 

who, when? policy!

3) 

have to ~~download~~ run packet filter code

10. Conclusions?

Other problems?

- large executables
- portability??

What was going on in OS world?

- OS ~~was~~ viewed as could not be changed, very complex
  - HP-UX, Ultrix, Solaris, Windows...

- Just let ~~people~~ interested developers change source code directly of Linux?

~~Make~~