Exokernel

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- Happy birthday, NCSA Mosaic
  - April 22, 1993
- Survey
  - Which OSs strike you as “tragic”? Why?
  - Who knows how to pronounce “quixotic”?
- Today: Exokernel
  - “Exterminate All Operating System Abstractions”
- ***No class Monday***

Overview

- The Exokernel worldview
  - Tragedy
  - Salvation
- My personal reaction to the Exokernel worldview

Tragedy

- The ***defining tragedy*** of the OS community
  - OS = hardware multiplexor
  - ***and*** OS = hardware abstractor
- OS abstraction is a ***quixotic goal***
  - Always-appropriate abstractions are ***impossible***
  - Always-efficient implementations are ***impossible***
- "The only way to win is not to play"

“Abstractions Considered Harmful”??

- No.
- The ***right*** abstractions are good.
  - But there is no ***single*** right abstraction
  - But each machine runs a ***single*** OS
    - A single process model
    - A single VM system
- One size ***cannot*** fit all
  - ***Ever***

What's the harm in trying?

- You say “quixotic goal” like it's a ***bad*** thing...
- Abstraction-heavy OSs are
  - Complex
  - Large
  - Unreliable
  - Hard to change
  - Slow

What's the harm in trying?

- You say “quixotic goal” like it's a ***bad*** thing...
- Abstraction-heavy OSs are
  - Complex
  - Large
  - Unreliable
  - Hard to change
  - Slow
- Not just ***one*** bad thing! ***Every*** bad thing!

Defining The Tragedy

- OS = any software you ***cannot avoid***
- Issue not “PL0 vs. PL4”
  - If you need to be administrator to install it, it's OS
  - Even if it runs in “user mode”
- Application software = anybody can avoid it

The Exokernel Thesis

- Q: Which jobs belong to the OS?
- A: Safe multiplexing of physical resources
  - Jobs which require the use of force
    - Timer interrupts force context switches
  - Preventing unfair initiation of force
    - Protecting my memory from your wild pointer
- ***Other jobs best done by other code structures***
  - Abstractions provided by ***voluntary*** use of libraries

What's ***wrong*** with OS abstractions?

- Complexity means bugs
- Complexity means inertia
- Complexity means slowness

Complexity means bugs

- If “virtual memory” means
  - Copy-on-write
  - Memory-mapped files
  - User-wired pages
  - Paging out parts of the OS kernel
- Then “virtual memory” will be buggy
  - For ***all*** processes
- (unless 15-412 students do the job)

Complexity means inertia

- Providing lots of fancy abstractions is ***hard***
  - Needs large, complex code
  - Large, complex code evolves slowly
- Everything depends on the OS
  - Changing the OS requires changing everything
  - Costly, slow
- Only illuminati can change the OS
  - “Linus doesn't scale”

Complexity means slowness

- Garbage collectors don't ***want*** dirty pages stored
  - If they're in the copied region of from-space
- Databases don't ***want*** dirty buffers written
  - If the transaction hasn't committed yet
- Databases don't ***want*** 1-block read-ahead
  - Bank withdrawals aren't sequential by account #
- A “free” OS optimization for one usage pattern...
  - ...is a ***mandatory OS slowdown*** for another pattern

The Horror is *Mandatory*

- There is ***only one*** file system
  - No other way to access the disk
- There is ***only one*** VM system
  - Only one page size, replacement policy, ...

“Virtual Machine Considered Harmful”

- The process model is bad
  - ***Every*** process model is bad
- CISC vs. RISC
  - Processors should provide ***basic instructions***
    - Load, store, copy register, add
  - Let ***compilers*** build them into abstractions
    - Procedure call, switch()
- “End-to-end Arguments In System Design”

Eliminate OS Abstractions

- Export hardware securely
- No machine-independent wrappers
- Abstraction-free kernel = ***exokernel***
  - All parts visible

Exokernel

- Safely allocate/deallocate/multiplex ...
  - memory ***pages***
  - CPU ***time slots***
  - disk ***sectors***
  - TLB ***slots*** (& address-space id #'s)
  - Interrupts & traps
  - DMA channels, bus bridges
  - I/O devices

The Real Hardware

- Real TLB, not abstract TLB
  - If version #13 has 32 entries and #14 has 64, deal
  - If version #17 had a broken reference bit, detect & deal
- Real memory, not abstract memory
  - You can ask for frames #31, #62
    - ...because you know they don't collide in ***this*** TLB
- You specify your own PTE entries
  - Don't forget to flush your TLB!!

Secure Multiplexing??

- ***Guards*** prevent evil
- PTEs you install map to ***your*** frames
- Packets you send are from ***your*** frames
  - Cannot “helpfully” free frame before complete
- Packets you receive are into ***your*** frames

“Is there an OS in the house?”

- Memory
- Polling
- CPU scheduling
- Packet transmission
- Packet filtering
- Packet buffering

Xok Memory

- Three OS data structures
  - per-process x86 page table
  - page access matrix
  - free page list
- Process can view its PTs
  - Check dirty, referenced bits for gc

Xok Memory

- Process requests changes
  - Simple, fast system call checks access
- Process may store a frame to disk
  - Or anywhere else
  - Then use frame for another page
- Process may maintain free-frame pool
  - It can/must handle its page faults

Abstract-OS Event Polling

- Wake me up when...
  - Client packet arrives, OR
  - Some client TCP connection can accept data
- Unix solution: select()/poll() system call
  - Works only on file descriptors
  - Expensive

Xok Event Polling

- Publish list of integers and comparisons
  - &socket->recv->count, &zero, GREATER
  - &socket->xmit->count, &16384, LESS
- Kernel generates, optimizes machine code
  - (pins pages)
- Scheduler runs per-process “runnable predicate”

Xok Packet Transmission

```
txpending = 1;
send(interface, iovec, &txpending);
```

- List of (address, length) pairs defines packet
- "txpending" integer decremented when done
  - might make process runnable

- Application must avoid overwriting packet

Xok Packet Filtering

- Application provides packet filter
- Kernel compiles into machine code
- Kernel checks for packet theft
  - This filter overlaps with an open filter, not yours
- Filtering != storage

Xok Packet Storage

- Process provides ring buffers in memory
- Kernel inserts packet
  - No room? Drop
- Kernel writes received-length field
  - Probably in receiver's “runnable predicate”

It's Weird. Is It *Good?*

- ExOS – ***voluntary*** POSIX emulation library
  - Provides file system, process semantics
  - Can run gcc, csh, etc.
- Simple socket-based HTTP server
  - 2X faster ExOS vs. OpenBSD
- Cheetah HTTP Server
  - Customized file system & TCP
  - ***3X-8X*** throughput of web servers on OpenBSD

Web Server Story

- Request/Response via TCP sockets
- Pre-fork()'d process pool for requests
- File data copied from disk to kernel to user
- File data copied user to kernel to network
- Slow
  - System calls block, fork() is slow
  - Checksum data before transmission
  - Memory-to-memory copies

Cheetah/Xok story

- Event loop instead of polling
  - Asleep until something is ready (disk, net)
  - Make it busy, sleep again
- Network retransmit buffers == file system cache
  - No duplication, no copy bandwidth
- Store ***TCP data checksums*** inside file
  - Computed offline when file is stored
  - ***Not*** computed for every transmission!

Eckhardt's Reactions

- 800% performance is exciting!
  - Wake-up call is good
  - Concepts & approach are a contribution
- There are issues

Objection: Multiplex != Allocate

- TLB slots exposed
  - How many for ***my*** process?
- CPU quantum expires
  - Who sets quantum length?
  - Which process is next?
  - “Next process” choice is rate-monotonic or not
    - Can't be rate-monotonic ***just for those who opt in***!

Objection: Multiplex != Allocate

- Disk interrupt!
  - Run newly-runnable process or just-interrupted one?
- These questions ***must be answered***
- ***Answers are mandatory abstractions***

Objection: Cooperative Multi-tasking?

- When kernel needs a frame
  - It ***asks*** a process to free one!
- Process should
    - Store page to disk (if neccesary)
    - Unmap page->frame
    - Free frame
- What if it doesn't???
- How can you distinguish “slow” from “no”???

Performance/Abstraction Issues

- (Identified in 2002 paper)
- “Runnable predicates” scale poorly
  - How to do better w/o OS-level abstractions???
- Run-time code generation brittle, hard to port
  - Inertia???
- No packet scheduler, so no connection fairness
  - Would be a ***mandatory abstraction***!

Applicability

- Do regular applications work well?
- Are genius programmers required?
  - Cheetah authors unusually high-powered...
- Is this really a general OS paradigm?
  - Is the tragedy over?

Summary

- “One size fits all” abstractions ***don't***.
- Abstraction mismatches are ***painful***.
- Multiplexing ***does not require abstraction***.
- Abstraction = box
  - Think “outside the box” for speed
  - ***Code*** outside the box?
    - Me?

Papers

- End-to-end Arguments In System Design
  - Saltzer, Reed, Clark: SOSP 5 (1981)
- Exterminate All Operating System Abstractions
  - Engler & Kaashoek: HotOS 5 (1995)
- Fast and Flexible Application-Level Networking on Exokernel Systems
  - Ganger, Engler, Kaashoek, Briceño, Hunt, Pinckney
  - ACM TOCS 20:1 (2002)