TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Paper Reading: *Application Performance and Flexibility on Exokernel Systems*

M. Frans Kaashoek, Dawson R. Engler et. al.

presented by Bjoern Doebel

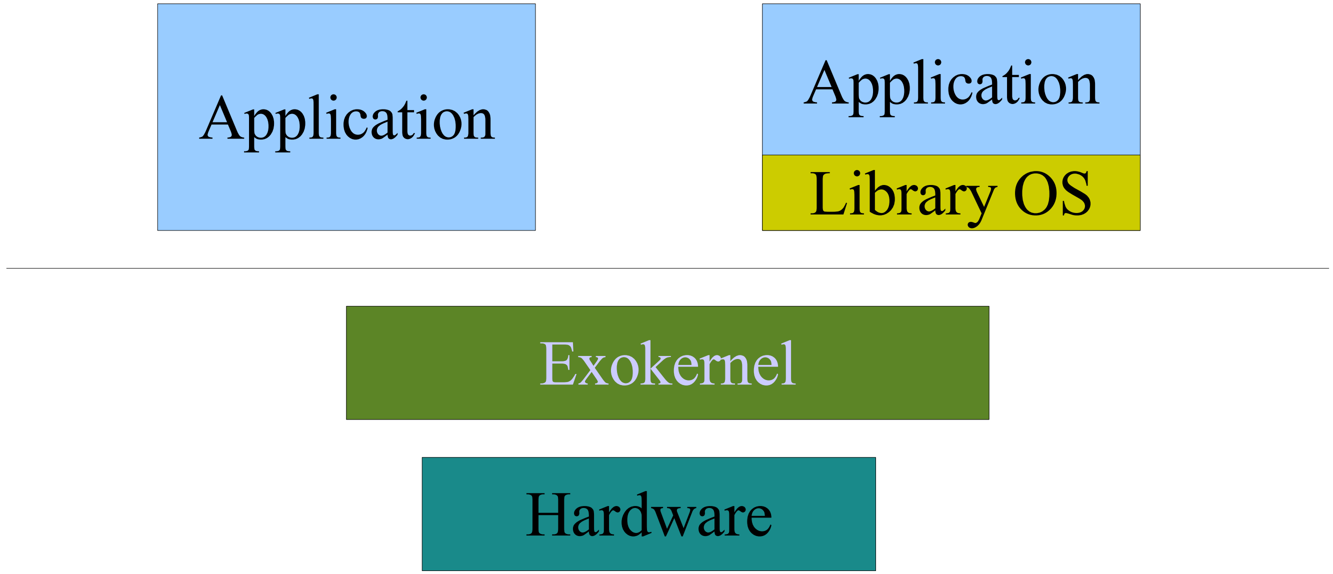
2006-11-01

Overview

- Introduction to
  - Exokernels
  - Library Operating Systems
- Xok and ExOS
- Multiplexing Stable Storage
- Performance
- Lessons learned

Exokernels

- “Exokernels protect resources, applications manage them.”



Exokernels

- Provide primitives at the lowest level of abstraction.
- Only manage resources to achieve protection.
  - allocation, revocation, sharing, ownership
- Support cooperative applications, but also handle evil ones.
- Expose physical names.
- Expose kernel information to applications.

Library Operating Systems

- provide higher-level OS abstractions
- linked with the application
- unprivileged, customizable
- application can “overwrite” specific library functionality
- need protection against other LibOSes:
  - software regions
  - hierarchically-named capabilities
  - wakeup predicates (downloaded code)
  - do not use locks for critical sections
- levels of trust (mutual, unidirectional, distrust)

The Xok Exokernel

- Kernel implemented for x86, similar to Aegis (MIPS)
- RR CPU Scheduling, explicit start/stop notifications
- Network multiplexing using packet filtering
- HW Page tables exposed through system calls
- explicit credentials (capabilities) needed to perform each system call

TECHNISCHE UNIVERSITÄT DRESDEN

The ExOS Library OS

- Targetted at providing abstractions similar to BSD
  - missing: paging, process swapping, process groups, windowing system
- Can run unmodified UNIX applications
- UNIX state mostly private to each instance of ExOS
- IPC:
  - pipes -> software regions
  - signals -> Xok IPC
  - sockets -> shared mem, network libs
- shared library without relocation overhead

XN: Multiplexing Stable Storage

- Problem 1: Disk operations need to be protected efficiently.
- Solution: Secure bindings
  - at bind time disk block is mapped to a physical memory page
  - FS server creates a capability for this page
  - everyone can now access the page using this capability

  -> Exokernel does not know about security policy, but enforces protection.

XN: Multiplexing Stable Storage (2)

- Problem 2:
  - Exokernel does not know about file systems and their metadata, but need to enforce protection.
- Solutions:
  - add capability for each disk block
    - way too slow
  - enhance blocks with application-specific data
    - not enough space
  - template-based metadata description
    - Exokernel should not define what block types exist.
  - Untrusted Deterministic Functions

XN: Untrusted Deterministic Functions (UDFs)

- Idea: Let the application / FS developer define how to extract protection information from the metadata.
- Templates specified using 3 UDFs:
  - owns(m): list of all disk blocks this metadata points to and their respective UDFs
  - acl(): boolean check if a certain modification can be performed given a capability
  - size(): size of metadata

XN: Ordering disk writes

- Disk writes need to be ordered to be consistent across crashes.
- Never reuse disk blocks before deleting all pointers to it.
  - reference counter for disk blocks
- Never create persistent pointers before initializing them.
  - “tainted” blocks must not be written to disk
- When moving blocks, never reset old pointers before new ones were set.
  - no need to enforce this

XN: Sharing blocks

- Applications can share disk blocks and map them to their own physical pages.
- Backing pages must be provided by application.
- Exokernel implements buffer cache registry to
  - cache established block-mem mappings
  - track state of mappings
- (De-)Allocation controlled by applications.
- Only block owners may issue write to disk.
  - Optimization: unowned disk blocks written by anyone (async. flush daemons)

The co-locating fast file system C-FFS

- Library File System on top of XN
- Uses downloaded code
- Four major additions:
  - Map Unix ACLs to exokernel capabilities
  - Enforce UNIX-specific semantics (e.g., unique file names in directories)
  - Enforce metadata consistency (blocks are not written to disk unless metadata is correct)
  - Keep track of file modification times etc.

Application performance

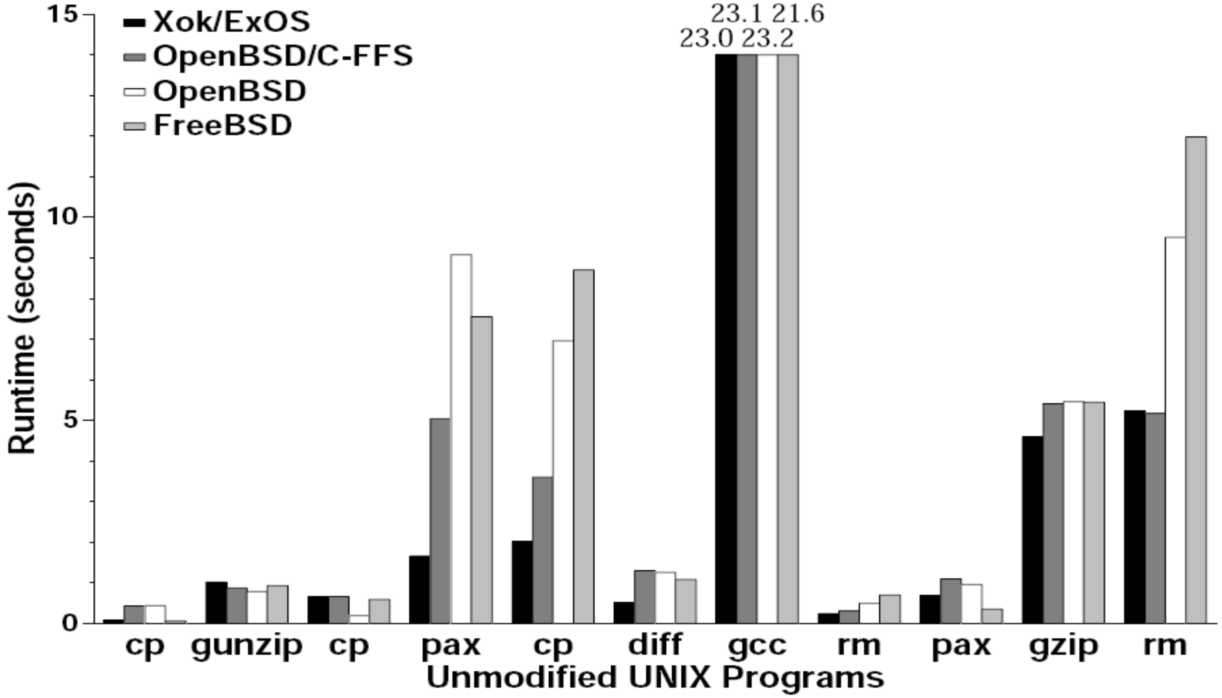
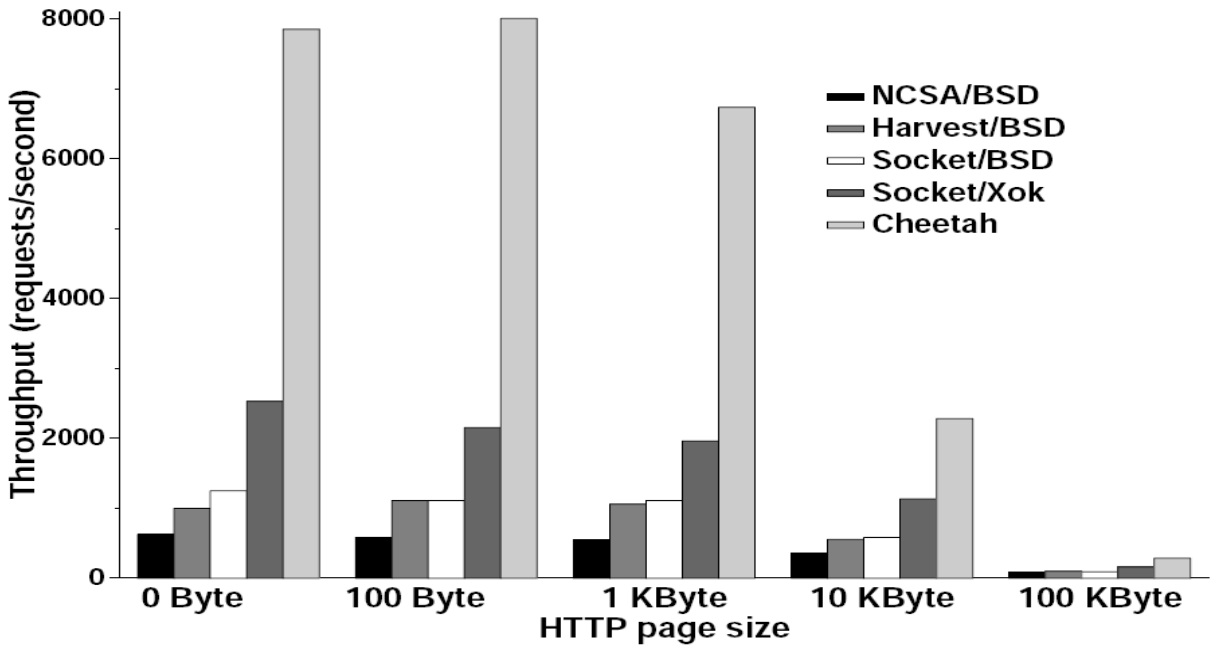


Figure 2: Performance of unmodified UNIX applications. Xok/ExOS and OpenBSD/C-FFS use a C-FFS file system while Free/OpenBSD use their native FFS file systems. Times are in seconds.

Application performance (2)

- Protection mechanisms add overhead, but real workloads are dominated by other things.
- Applications custom-tailored for ExOS can outperform legacy applications.



Exokernels – Pros and Cons

- Pros
  - Exposing internal kernel structures
  - Libraries are simpler than kernels
- Cons
  - Interface design is difficult
  - Information loss
  - Self-paging LibOSes proven difficult

Design lessons learned

- Provide space for app data in kernel structures
- Fast apps are necessarily good in microbenchmarks
- Need for inexpensive critical sections
- User-level page tables are complex.
- Downloaded code is powerful.

In the end

I'm done.