Application Performance and Flexibility On Exokernel Systems

–M. F. Kaashoek, D.R. Engler, G.R. Ganger et. al.
Presented by Henry Monti
Exokernel

- What is an Exokernel?
- What are its advantages?
- What are its disadvantages?
- Performance
- Conclusions
The Problem

- Traditional operating systems provide both protection and resource management
- Will protect processes address space, and manages the systems virtual memory
- Provides abstractions such as processes, files, pipes, sockets, etc
- Problems with this?
Exokernel's Solution

• The problem is that OS designers must try and accommodate every possible use of abstractions, and choose general solutions for resource management

• This causes the performance of applications to suffer, since they are restricted to general abstractions, interfaces, and management

• Research has shown that more application control of resources yields better performance
(Separation vs. Protection)

- An exokernel solves these problems by separating protection from management
- Creates idea of LibOS, which is like a virtual machine with out isolation
- Moves abstractions (processes, files, virtual memory, filesystems), to the user space inside of LibOSes
- Provides a low level interface as close to the hardware as possible
General Design Principles

- Exposes as much as possible of the hardware, most of the time using hardware names, allowing libOSes access to things like individual data blocks
- Provides methods to securely bind to resources, revoke resources, and an abort protocol that the kernel itself uses
Exokernel Design

Diagram showing the structure of an exokernel environment. The diagram includes:

- **Mosaic** component with **WWW**, **POSIX**, and **TCP**.
- **Applications** section with **Barnes-Hut**.
- **Library operating systems** section with **DSM**, **IPC**, **VM**, and **Traps**.
- **Secure bindings** connecting the exokernel to hardware components:
  - **Exokernel**
  - **Hardware** components: Frame buffer, TLB, Network, Memory, Disk.
Extensibility

• Applications can link to LibOS as shared library, and since LibOSes are unprivileged they can be changed, added to the system or upgraded.

• This gives the application programmer the ability to specialize every part of their application.

• In order to provide many OS abstractions, privileged instructions must be exported to LibOSes.
Virtual Memory

• Virtual Memory is implemented at user level
• In the Linux kernel, everything is handled inside of the kernel, the kernel maintains a list of vmas, and signals, adds and removes them as necessary
• In an exokernel, the kernel would encounter a page fault and not know what action to take, so it would inform the application, which maintains its own pages
Virtual Memory

• The application decides what to do about the page fault, and if necessary asks the exokernel to allocate the new page, and then to store the mapping of the page (virtual)

• The exokernel would just guarantee that application received the info, and once the application made a decision, that it had rights to allocate, access, free, move a data page frame (physical)
Challenge

- While distributed management can have its advantages in many cases, there are some where it doesn't make as much sense.
- Sharing resources in an exokernel environment can be very difficult.
- A good example of this is a hard drive, which is a shared system resource.
- In a typical operating system, the kernel has knowledge of the file system so it can easily provide protection, and multiplex the resource.
Challenge

• An exokernel would not have specific knowledge of the filesystem/meta data, or layout, but instead protect disk block access.

• In fact, an exokernel would likely have several libOSes each running multiple libFSes, accessing many different file types.

• In order to deal with this challenge, the authors spend significant time showing that an exokernel can successfully multiplex the disk.
Multiplexing Stable Storage

- Creating new file formats should be lightweight and easy, requiring no special permission
- Should allow multiple libFSes to safely share files at the meta data and block level
- Must be as close to the raw hardware as possible
- Facilitate cache sharing across LibFSs
XN

- To deal with these issues the authors introduce XN as way to provide protection with as little management as possible
- Main goal is to determine access rights of a given principal
- To create new file types, the exokernel provides udfs (untrusted deterministic functions), which are method of specifying file meta data on the fly
UDFs

• Assembly like language that kernel understands, which allows libFSes to specify the meta data type

• LibFS would create a UDF for each meta data type, direct block, indirect block, doubly indirect

• Example, in order to allocate block b, on metadata block m, runs udf(m), makes the proposed modification on m', then runs udf(m'), to make sure the udf(m') = udf(m) + b
G. Back S07, CS3204 project 4 slides
**XN**

- The Authors define the following rules to allow application control and protected sharing, XN must:
  - To prevent unauthorized access every disk data operation must be guarded
  - Must be able to determine the access rights a libFS has to a data block
  - Must ensure consistency across system crashes
XN

- To allow protected sharing of file system state by mutually distrustful LibFSes, there are additional rules that must be met
- Coherent caching of disk blocks, many potential problems with libOS specific caches
- Atomic meta-data updates
- Well formed updates
XN

- XN has a global buffer cache registry, applications contain the data pages themselves, while the registry contains mappings
- Provides a method of protected sharing, and provides an easy method to enforce cache coherency
What are the Advantages?

- Why move all of these abstractions to LibOSes?
- Performance.
- Better performance can be achieved in two ways, applications can now manage their own resources
- Second, the exokernel itself can be implemented small, with fast primitives
Exokernel Advantages

• Since LibOSes run at the user level, if an error occurs inside of one only the application will calling the libOS will be affected

• Since more than one LibOS can be created, different operating system interfaces can be implemented, and unmodified programs can be run ex. BSD on an exokernel

• LibOSes provide and interface to expand and create new operating system abstractions, in a relatively pain free way
Disadvantages

• Exokernel interfaces are hard to write, one must strive for transparency and while maintaining protection

• Global performance:

• While the tests the performed seem to indicate that their exokernel capable of providing good global performance, the architecture seems like it could still be open to denial of service attacks
Other Problems

- Information Loss, “if virtual memory and the filesystem are completely at the application level, the exokernel may be unable to distinguish pages used to cache disk blocks and pages used for virtual memory”
Xok/ExOS

- Xok is their implementation of the exokernel architecture for the X86 processor
- They have also implemented exokernels on MIPS DEC workstations and other processors
- Since an exokernel is so tightly tied to the underlying hardware, each architecture must be handled in different ways
- Example: Page table structure is defined in hardware on X86, so unlike MIPS applications can't override and create their own
ExOS

- ExOs is a libOS that supports the operating system abstractions of 4.4 BSD,
- Implemented as library so it can be modified
- Implements the behavior, processes, interprocess communication, file descriptors of a typical UNIX system
Performance

- Performed experiments on both standard UNIX programs and applications written specifically for XoK
- Ran a series of bench marks on XoK, FreeBSB and OpenBSD. Xok, the exokernel either provides similar performance or significantly better performance
- Exokernel Xok and LibOS ExOS are “free”, they don't create a large additional penalty for standard UNIX programs
Exokernel Specific Applications

- Binary Emulator, emulates BSD system calls so programs without available source code can be used.
- Emulator and emulated program run in the same address space, so there is only a few percent reduction vs. running directly on Xok.
- Emulation actually runs faster on Xok than natively in BSD, (system calls vs. procedure calls).
Exokernel Specific Applications

- XCP, an efficient file copying program
- Improves performances with disk scheduling, by never having the CPU “touch” the data, with asynchronous IO
- Compared to “cp” on Xok, xcp is 3 times as fast
Exokernel Specific Applications

- Cheetah Web Server, a fast web server that utilizes Xok
- Uses many optimizations such as co-locating html files on disk, so that all of the files for one web page are grouped closely together
- Cheetahs outperforms web servers that use traditional interfaces, performance 8x faster than in BSD
Conclusions

- Exokernels seek to expose as much hardware information as possible without violating protection rules.
- At least some current operating systems could be implemented as an libOS with minimal performance impact.
- Program performance can dramatically improve when they take advantage of the exokernel features.
Conclusions

• While exokernels can improve performance, they suffer from problems, from creating the exokernel interface to information loss.
Evaluation

• Exokernels are ideally suited to situations where performance is must, and unoptimized operating system abstractions are constraining performance, web servers, data bases

• Seems like an excellent research tool for implementing and testing new operating system abstractions and ideas
Evaluation

- Problems...

- Global performance not thoroughly evaluated, could potentially be more vulnerable to denial of service attacks, which would invalidate the performance gains

- After nearly 15 years after its initial introduction there are only a handful exokernel systems, only in the hands of researchers, hasn't made it to the mainstream