Application Performance and Flexibility on Exokernel Systems

Kaashoek, Engler, Ganger, Briceno, Hunt, Mazieres, Pinckney, Grimm, Jannotti, Mackenzie

CS5204 – Operating Systems

Christopher J. Goddard

11-02-2005

1

- Traditional Operating Systems
  - Rely on kernel and privileged servers to manage system resources
- Unprivileged applications required to use interfaces of privileged software
  - Applications cannot do what they want
    - May run slowly
    - Or, cannot be implemented at all

2

- An “interface for all”
  - Must consider possible needs and all ways it could be used
  - Must consider trade-offs
- Authors' Position
  - This sort of anticipation is:
    - Prone to mistakes
    - Restrictive
    - Infeasible

3

- Separate protection from management
  - Kernel protects resources
  - Applications manage their own resources
  - End-to-end argument
- Example: Application manages its disk-block cache and kernel allows cached pages to be shared securely between applications
- Result: Exokernel approach

4

- Of course, not all applications need (or want) to their own customized resource management scheme
- Need for traditional abstractions (shells, file utilities, other UNIX abstractions)
- Library operating systems (libOSes) provide these abstractions
  - In this way the application need not communicate with exokernel directly

5

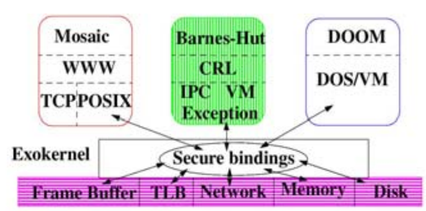

Image from SOSP '95 Exokernel Talk (http://pdos.csail.mit.edu/~engler/exo-sosp-talk.ps)

6

- Step by step integration of new OS features
- New functionality can be distributed with applications
- Any skilled programmer can create a standalone libOS without having to modify rest of system
- There are many more application authors than there are operating system developers (weak)

7

- Goal of exokernel: to give applications more control
- Challenge:
  - To provide extensibility to applications, allowing them to exploit performance
  - Also, to provide a base for a general purpose (well rounded) system
    - Ideal: Operating systems built on top of exokernel perform just as well as, or better than, current operating systems

8

- Separate protection from management
  - Exokernel is only allowed management necessary for protection
- Expose Allocation
  - Applications allocate resources explicitly
- Expose Names
  - Use physical names whenever possible
  - Capture useful information

9

- Expose Revocation
  - Revocation policies exposed to applications
  - Allowed to decide which resource instance to give up
  - Each application has control over its physical resources
- Expose Information
  - All system information is exposed to applications
  - Examples:
    - Application knows number of hardware network buffers
    - Application knows which pages cache file blocks

10

- Kernel protection of high level abstractions
  - Files = metadata, disk blocks, buffer cache pages
  - Want access control on high level (permissions)
  - But exokernels allow access to low level resources
- Main challenge in designing exokernels
  - Discover kernel interfaces that provide high level access control
  - Do not require a specific implementation
  - Do not limit application control over low level resources

11

- Advantages of library implementations:
  - Can trust applications that are using them
  - Need not protect against malicious use
- Disadvantages:
  - Cannot trust other libOSes that have access to a particular resource
  - Guarantees regarding invariants must take into consideration other processes that have access to the resource
  - What level of trust should the libOS place on other processes?

12

- Mutual Trust
  - Common case
  - Example: UNIX programs run by the same user trust each other
  - Similarly, when two exokernel processes can write each others' memory, libOSes can trust each other
- Unidirectional Trust
  - Two processes share resources – one trusts the other but trust is not mutual
  - Example: Network Servers – Privileged process accepts connection, forks, performs as user

13

- Mutual Distrust
  - Two processes share high level abstractions but distrust each other
  - Example: Two unrelated processes communicate over UNIX socket but neither has trust
  - libOSes must reasonably and defensively interpret all actions of the other process
    - Example: Socket write larger than buffer interpreted as end of file
  - Mutual distrust is infrequent

14

- Exokernel must multiplex disks across multiple library file systems (libFSes) which are contained within each libOS
- libFSes can define new file types with different metadata characteristics
- Exokernel must give libFSes control over hardware while still protecting files from unauthorized access

15

- Exokernel has a difficult job
  - Must multiplex disk but cannot rely on simple techniques to do so
- Exokernel must follow four requirements:
  - New file formats should be simple/lightweight
  - libFSes should be able to share files at raw disk block and metadata level
  - Storage must be efficient (close to raw hardware performance)
  - Cache sharing between libFSes

16

- Stable storage is difficult to implement
  - XN is the authors' *fourth design*
- *XN's main purpose: Efficiently determine access rights of a principal to a disk block*
  - *Prevent a user from claiming another user's disk blocks as part of their own files*
- *Conventional OS: Easy – knows metadata format*
- *Exokernel: Hard – application defined metadata*

17

- XN uses *untrusted deterministic functions (UDFs)*
  - *These are specific to each file type and translate metadata*
  - *Used to translate metadata into a simple form for the kernel to use*
  - *UDFs can be installed by a developer to define new metadata formats*
- *UDFs enable the kernel to handle metadata formatting without actually having to understand the format itself*

18

- The *co-locating fast file system (C-FFS) is a UNIX-like libFS*
- Access Control
  - Provides UNIX style access control (uids, gids, etc.)
- Well Formed Updates
  - Supplies UNIX specific file semantics (legal filenames)
- Atomicity
  - Performs locking (ensures that data is recoverable)
- Implicit Updates
  - Certain state transitions imply actions (mod times)

- ExOS: A libOS that provides many 4.4BSD abstractions
  - Runs many unmodified UNIX applications
    - Most shells, file utilities (grep, ls, etc.), and network programs (telnetd, ftp, etc.)
  - Missing functionality
    - Full paging, process groups, windowing system
    - Authors note that there is no reason why these cannot be implemented; they just did not have time to implement or port them

- Goals
  - Simplicity
  - Flexibility
- Entirely library based, so applications can override any feature
- Any functionality of ExOS can be replaced by application-specific code

- Cheetah HTTP Server
- Merged File Cache and Retransmission Pool
  - Uses precomputed file checksums which are stored in each file
  - No in memory data touching by CPU
  - Transmission/retransmission(s) directly from file cache
- HTML-based File Grouping
  - HTML document and associated files co-located on disk

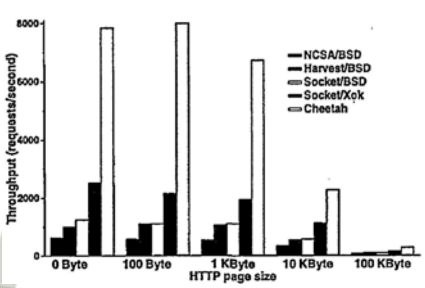

Image from *Application Performance and Flexibility on Exokernel Systems*

- Libraries are simpler than kernels
  - Over many iterations, “edit, compile, debug” much faster than the “edit, compile, reboot, debug” associated with traditional kernels
  - Libraries easier to debug due to isolation from rest of system
- But... exokernel interface not easy to design
  - Exokernel must export low level interfaces but also offer protection
  - Several iterations required to develop a substantial interface

- Exokernel is a radical change in traditional kernel design
- Using libraries to implement file systems and operating systems gives applications more power
  - Applications can take advantage of performance
  - Good foundation for a multipurpose operating system
- Libraries themselves not always so easy to implement
  - “Large implementor base” is questionable

25

- Questions or comments?

26

27

- Virtual machine monitor
  - Privileged
  - Isolates less privileged applications in emulated copies of hardware
- However, emulation hides information
- VMMs confine processes to virtual machines, whereas exokernels give applications access to libOSes without compromising a single view of the machine

28

- Each UDF is stored (on disk) as a template which corresponds to a metadata format
  - Example: A UNIX file system would have templates for data blocks, inodes, indirect blocks, etc.
  - A template cannot be changed after it is specified
- Each template contains at least one non-deterministic function (*owns-udf)*
- *Functions are written in a pseudo-RISC assembly language (checked by kernel for determinacy)*

29

- So, interpreted UDFs allow libFSes to track access rights
- XN does not need to understand
- XN just verifies that block ownership is tracked correctly

30

- The *co-locating fast file system (C-FFS) is a UNIX-like libFS*
- *Since XN just provides basic file system integrity guarantees, more specific invariants may be needed*
  - *Example: UNIX file systems guarantee uniqueness of filenames within a directory*
- *Provides four additions to XN's protection policies*

31

- Exokernel: Xok
- Multiplexes physical resources
- Virtual memory abstractions at application level
  - Exposes hardware capabilities
  - Exposes kernel data structures
  - Low level interface allows for paging at the application level
    - Paging can be done from disk or over the network
    - Allows for page transformations – compression, digital signatures and verification, or encryption

32

- Issue: Process might want to sleep until a certain condition is true
  - Difficult with exokernels since applications handle most of the OS functionality
- Solution: Wakeup predicates
  - Wakeup predicates are injected into kernel by application
  - Boolean expressions used by application to wake when state of system changes

33

- Simple, yet powerful
  - Due to application control
- Example:
  - To wait for a disk block to finish being paged in, an application can use a wakeup predicate to wait for the block's state to change from “in transit” to “resident”
- Example:
  - Webserver

34