Exokernel:

An Operating System Architecture for Application-Level Resource Management

Dawson R. Engler,
M. Frans Kaashoek, and
James O'Tool Jr.
M.I.T Laboratory for Computer Science
Cambridge, MA 02139, U.S.A

Presented by Jeffrey J. Weston

Abstractions are Bad

- Problems with monolithic kernel abstractions:
  - Applications are unable to use domain specific optimizations.
  - Changes to the implementation of existing abstractions is discouraged.
  - Restricts flexibility of application builders to add new abstractions since they must be added on top of existing abstractions.

Abstractions are Bad

- Problems with micro-kernel abstractions:
  - Untrusted applications are unable to define specialized IPC primitives because virtual memory and message passing services are implemented by the kernel and trusted servers.
  - Many abstractions, such as page table structures and process abstractions, cannot be modified in micro-kernels.
  - Many hardware resources in micro-kernels are encapsulated in heavy-weight servers that cannot be bypassed or modified.

What is an Exokernel?

- Similar to micro-kernel; limited functionality is built into the kernel.
- User applications have low-level access to hardware resources.
- Kernel provides support for applications to manage their own access to hardware resources.

Exokernel Design

- Exokernel consists of a thin layer that exports and multiplexes physical resources through a set of low-level primitives.
- Applications, using the low-level exokernel interface, implement higher level abstractions that can better meet the performance and functionality needs of the application.

Exokernel Design

- Exokernel uses three techniques to control access to machine resources:
  - Secure Bindings: Applications can securely bind to machine resources.
  - Visible Revocation: Applications can participate in the resource revocation protocol.
  - Abort Protocol: Exokernel can break the secure binding of non-responsive applications by force.

Resource Sharing: Processor

- Application can decide duration and frequency of their time slices.
- When a time slice is over, application event handler is called, allowing application to customize the context switch.
- Time taken during the event handler is borrowed from the next time slice.
- If event handler takes too long, a context switch is forced by the exokernel.

Application-Specific Handlers

- Exokernel provides support to download code to the kernel, called application-specific handlers (ASH).
- This provides two advantages:
  - Eliminates kernel crossings.
  - The execution time of downloaded code is bounded and can thus be run in cases where the application cannot be scheduled.

Application-Specific Handlers

- In the context of networking, ASHs are untrusted user-level message-handlers that are downloaded into the kernel.
- Made safe by:
  - Code Inspection
  - Sandboxing
- Message handlers are executed when a message arrives.

Application-Specific Handlers

- Abilities of ASH message-handlers:
  - Control where messages are copied in memory, eliminating intermediate copies.
  - Integrated layer processing, such as checksums.
  - Message initiation, allowing for low-level message replies.
  - Control initiation.

Exokernel Implementation

- Aegis, an exokernel, was built using the exokernel architecture.
- Aegis exports the following resources:
  - Processor
  - Physical Memory
  - TLB
  - Exceptions
  - Interrupts
  - Network

Exokernel Implementation

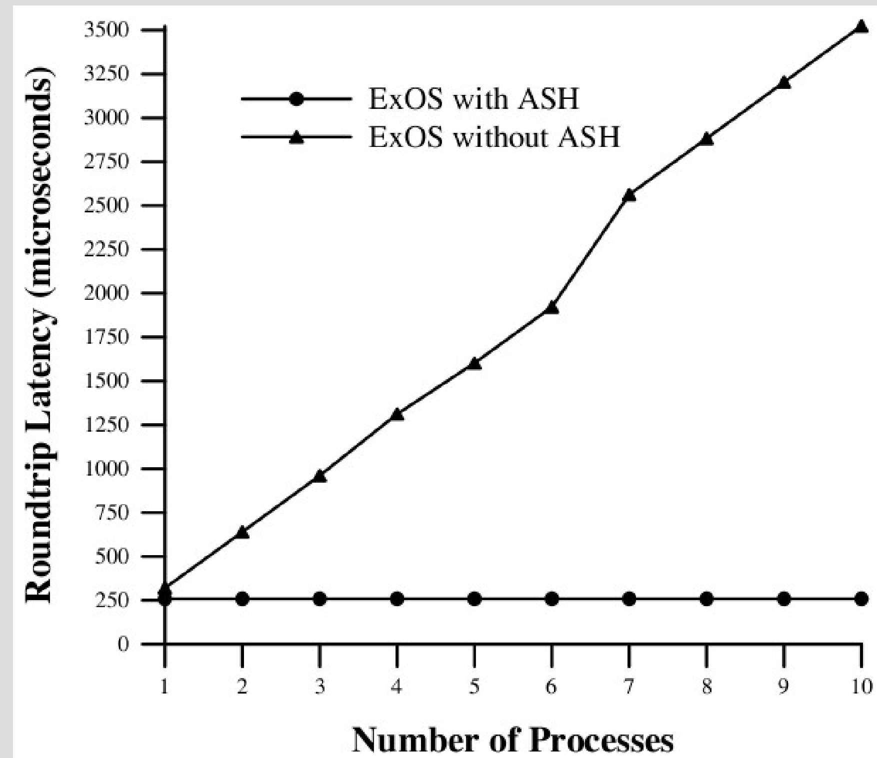
- ExOS is a library operating system built to run on top of Aegis.
- ExOS implements the following abstractions:
  - Processes
  - Virtual Memory
  - User-Level Exceptions
  - Various Interprocess Abstractions
  - Several Network Protocols

Performance: ASHs

Machine	OS	Roundtrip latency
DEC5000/125	ExOS/ASH	259
DEC5000/125	ExOS	320
DEC5000/125	Ultrix	3400
DEC5000/200	Ultrix/FRPC	340

Roundtrip latency of a 60-byte packet over Ethernet using ExOS with ASHs, ExOS without ASHs, Ultrix, and FRPC; times are in microseconds.

Performance: ASHs



Average round-trip latency with increasing number of active processes on receiver.

Performance: IPC

Machine	OS	pipe	pipe'	shm	lrpc
DEC2100	Ultrix	326.0	n/a	187.0	n/a
DEC2100	ExOS	30.9	24.8	12.4	13.9
DEC3100	Ultrix	243.0	n/a	139.0	n/a
DEC3100	ExOS	22.6	18.6	9.3	10.4
DEC5000	Ultrix	199.0	n/a	118.0	n/a
DEC5000	ExOS	14.2	10.7	5.7	6.3

Time for IPC using pipes, shared memory, and LRPC on ExOS and Ultrix; times are in microseconds. Pipe and shared memory are unidirectional, while LRPC is bidirectional.

Performance: Virtual Memory

Machine	OS	dirty	prot1	prot100	unprot100	trap	appel1	appel2
DEC2100	Ultrix	n/a	51.6	175.0	175.0	240.0	383.0	335.0
DEC2100	ExOS	17.5	32.5	213.0	275.0	13.9	74.4	45.9
DEC3100	Ultrix	n/a	39.0	133.0	133.0	185.0	302.0	267.0
DEC3100	ExOS	13.1	24.4	156.0	206.0	10.1	55.0	34.0
DEC5000	Ultrix	n/a	32.0	102.0	102.0	161.0	262.0	232.0
DEC5000	ExOS	9.8	16.9	109.0	143.0	4.8	34.0	22.0

Time to perform virtual memory operations on ExOS and Ultrix; times are in microseconds. The times for appel1 and appel2 are per page.

Conclusion

- Low-level primitives can be implemented very efficiently.
- Low-level multiplexing of hardware resources can be handled efficiently.
- Traditional operating system abstractions can be implemented efficiently at the application level.
- Applications can make special purpose abstractions by merely modifying a library.

Acknowledgement

Tables and figures were adapted from "Exokernel: An Operating System Architecture for Application-Level Resource Management" by Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.