Exokernel

– An OS Architecture for Application-Level Resource Managment

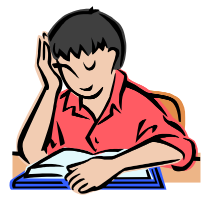
Shufang Wu
shufang.wu@ieee.org
Thursday, March 10, 2005

Agenda

- Paper Description (1 Slide)
- What is the Observed Problem? (1 Slide)
- What is the Proposed Solution? (10 Slides)
- How is the Solution? (15 Slides)
- What We Learned (1 Slide)
- References
- **Q & A**

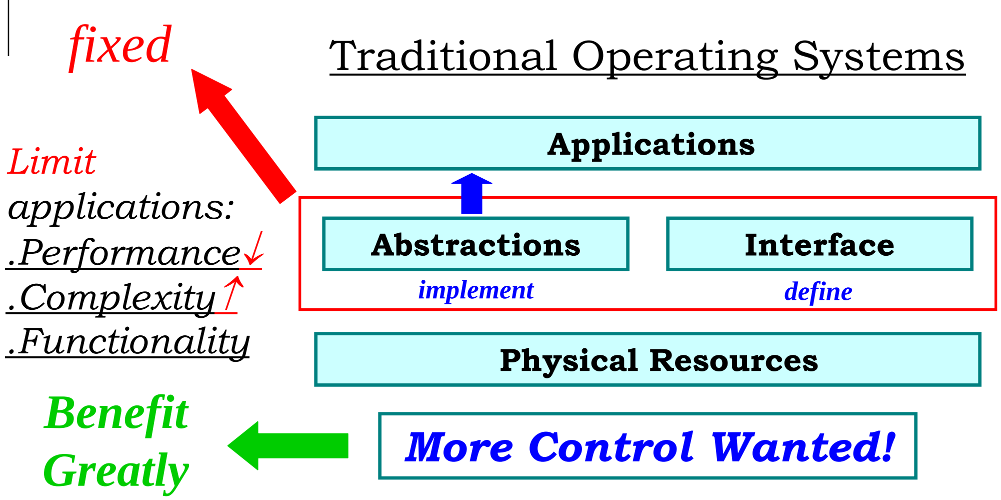
Paper Description

- Pages: 16 (including 2 pages of references)
- Two-columns
- Main text:
  - Font size: 9 => 9



Too much! I have to choose some.

Problem



Solution

Proposed Operating System Architecture

Application Level Resource Management

Solution – Exokernel

- Applications Know Better Than OS
- A Simple, Thin veneer:
  - Multiplex and export physical resources securely through a set of primitives
- Library OS:
  - Simpler and more specialized
  - Portability and compatibility
  - Simplified by modular design

Solution – Design (1133)

- ***One*** Goal
  - Give applications more freedom in managing
- ***One*** Way
  - Separate protection from management
- ***Three*** Tasks
  - Track ownership
  - Ensure protection
  - Revoke access
- ***Three*** Techniques
  - Secure binding
  - Visible revocation
  - Abort protocol

Solution – Design Principles

- Securely expose hardware
  - The central tenet of the architecture
  - All privileged instructions, hardware DMA capabilities, and machine resources
- Expose allocation
  - Allow to request specific physical resources
- Expose Names
  - Remove a level of indirection: Translation
- Expose Revocation
  - Allow to relinquish

Solution – Design Policy

- Exokernel Hands over
  - Resource policy decisions to applications/library OS
- Exokernal must include policy to
  - Arbitrate between competing applications/library OS
  - At this point, no different from traditional kernels

Solution – Secure Bindings

- Is A Protection Mechanism
  - decouple authorization from the use
- Can Improve Performance
  - Protection checks expressed in simple ops
  - Perform authorization only at bind time
- Primitives can be implemented in h/w or s/w
  - Hardware mechanisms
  - Software caching
  - Downloading application code

Secure Bindings – Examples

- Multiplexing Physical Memory
  - Using *self-authenticating capability* and *address translation hardware*
  - *To ensure protection*: guards access by requiring to present the capability
  - *To break*: change capability and free resource
- Multiplexing the Network
  - A software support is provided by *packet filters*
  - Application code, *filters,* is downloaded into kernel

Secure Bindings – Examples

- Application-specific Safe Handlers (***ASH***)
  - An example of downloading code
  - Downloaded into kernel to initiate a message
  - Associated with a packet filter
  - Runs on package reception

Solution – Visible Revocation

- Way to Reclaim and Break
- Compared to Invisible Revocation
- Can Guide De-allocation and Have Knowledge
- A Requirement of Physical Naming

Solution – Abort Protocol

- Exokernel Takes Back Resources "*By Force*"
- Break All Bindings and Inform
- *Repossession Vector*
  - Record the forced loss of resource
- *Repossession Exception*

How's the Solution? (15 – 1)

- Prototype
  - *Aegis* (exokernel), and *ExOS* (library OS)
- *Aegis*
  - CPU, MEM, Exception, TLB, Interrupt, NI
- *ExOS*
  - Process, VM, User-level exceptions, Interprocess abstractions, Network protocols
  - Extensibility

How's the Solution? (15 – 2)

Machine	Processor	SPEC rating	MIPS
DEC2100 (12.5 MHz)	R2000	8.7 SPECint89	~ 11
DEC3100 (16.67 MHz)	R3000	11.8 SPECint89	~ 15
DEC5000/125 (25 MHz)	R3000	16.1 SPECint92	~ 25

Table 1: Experimental platforms.

How's the Solution? (15 – 3)

- Test Four Hypotheses
  - Exokernel can be very *efficient*
  - Low-level, secure multiplexing of hardware resources can be implemented efficiently
  - Traditional OS abstractions can be implemented efficiently at application level
  - Applications can create special-purpose implementation of these abstractions

How's the Solution? (15 – 4)

***Aegis*: As an Exokernel**

System call	Description
`Yield`	Yield processor to named process
`Scall`	Synchronous protected control transfer
`Acall`	Asynchronous protected control transfer
`Alloc`	Allocation of resources (*e.g.*, physical page)
`Dealloc`	Deallocation of resources

Table 2: A subset of the Aegis system call interface.

Primitive operations	Description
`TLBwr`	Insert mapping into TLB
`FPUmod`	Enable/disable FPU
`CIDswitch`	Install context identifier
`TLBvadelete`	Delete virtual address from TLB

Table 3: A sample of Aegis's primitive operations.

How's the Solution? (15 – 5)

Aegis: Base Costs

Machine	OS	Procedure call	Syscall (getpid)
DEC2100	Ultrix	0.57	32.2
DEC2100	Aegis	0.56	3.2 / 4.7
DEC3100	Ultrix	0.42	33.7
DEC3100	Aegis	0.42	2.9 / 3.5
DEC5000	Ultrix	0.28	21.3
DEC5000	Aegis	0.28	1.6 / 2.3

Table 4: Time to perform null procedure and system calls. Two numbers are listed for Aegis's system calls: the first for system calls that do not use a stack, the second for those that do. Times are in microseconds.

How's the Solution? (15 – 6)

Aegis: Exceptions

Machine	OS	unalign	overflow	coproc	prot
DEC2100	Ultrix	n/a	208.0	n/a	238.0
DEC2100	Aegis	2.8	2.8	2.8	3.0
DEC3100	Ultrix	n/a	151.0	n/a	177.0
DEC3100	Aegis	2.1	2.1	2.1	2.3
DEC5000	Ultrix	n/a	130.0	n/a	154.0
DEC5000	Aegis	1.5	1.5	1.5	1.5

Table 5: Time to dispatch an exception in Aegis and Ultrix; times are in microseconds.

How's the Solution? (15 – 7)

Aegis**:** **providing protected control transfer as substrate for efficient IPC implementation**

OS	Machine	MHz	Transfer cost
Aegis	DEC2100	12.5MHz	2.9
Aegis	DEC3100	16.67MHz	2.2
Aegis	DEC5000	25MHz	1.4
L3	486	50MHz	9.3 (normalized)

Table 6: Time to perform a (unidirectional) protected control transfer; times are in microseconds.

L3: **the fastest published result.**

How's the Solution? (15 – 8)

Aegis**:** **using Dynamic Packet Filter**

Filter	Cold Cache	Warm Cache
MPF	71.0	35.0
PATHFINDER	39.0	19.0
DPF	7.5	1.5

Table 7: Time on a DEC5000/200 to classify TCP/IP headers destined for one of ten TCP/IP filters; times are in microseconds.

MPF: **a widely used packet filter engine.**

PATHFINDER: **fastest packet filter engine.**

How's the Solution? (15 – 9)

Conclusion for *Aegis*

An exokernel
can
be implemented
***efficiently*!**

How's the Solution? (15 – 10)

***ExOS*:**
Manage OS abstractions at application level

Focus on:
- IPC Abstractions
- Application-level Virtual Memory
- Remote Communication

How's the Solution? (15 – 11)

ExOS: IPC Abstractions

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

Table 8: 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.

How's the Solution? (15 – 12)

ExOS: Virtual Memory measured by matrix multiplication

Machine	OS	matrix
DEC2100	Ultrix	7.1
DEC2100	ExOS	7.0
DEC3100	Ultrix	5.2
DEC3100	ExOS	5.2
DEC5000	Ultrix	3.8
DEC5000	ExOS	3.7

Table 9: Time to perform a 150x150 matrix multiplication; time in seconds.

How's the Solution? (15 – 13)

ExOS: Virtual Memory On Seven Experiments of Particular Interest

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

Table 10: Time to perform virtual memory operations on ExOS and Ultrix; times are in microseconds. The times for **appel1** and **appel2** are per page.

How's the Solution? (15 – 14)

ExOS: Remote Communication

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

Table 11: Roundtrip latency of a 60-byte packet over Ethernet using ExOS with ASHs, ExOS without ASHs, Ultrix, and FRPC; times are in microseconds.

FRPC: fastest RPC on comparable hardware.

How's the Solution? (15 – 15)

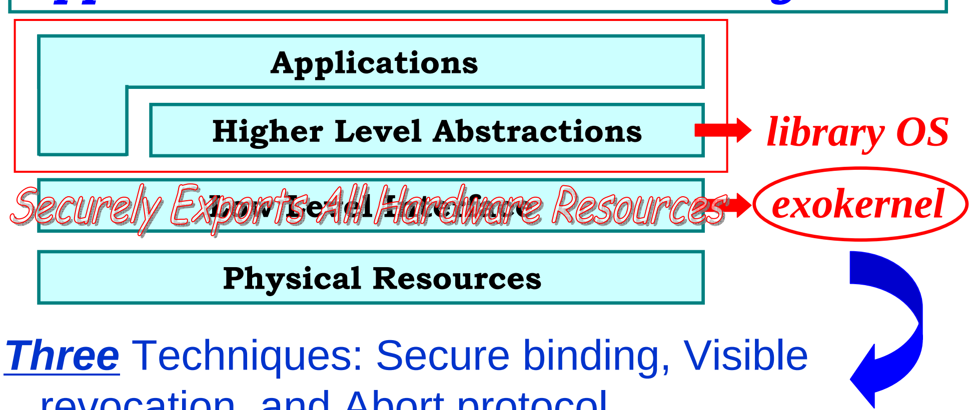
***ExOS*: No Conclusion in Paper☺**

Based on the results of these experiments, we *conclude* that:

The exokernel architecture is a *viable* structure for *high-performance, extensible* operating systems.

What We Learned?

Application Level Resource Management



Three Techniques: Secure binding, Visible revocation, and Abort protocol

References

1. Dawson R. Engler, M. Frans Kaashoek, and James O’Toole Jr., “Exokernel: An Operating System Architecture for Application-Level Resource Management", Proc. Of 15th Symposium on Operating System Principles, December 1995, pp. 251-266

Thank You!

QUESTION ?