Extensible Operating Systems

Shrutarshi Basu

Why Extensible Operating Systems?

- Appel and Li — general purpose VM primitives impact peristent stores, GC and distributed shared memory
- Cao et al. — application control over file caching reduces running time by 45%.
- Harty and Cheriton and Krueger et al. — app-specific VM policies increase performance
- Stonebraker — file-system implementations affect database performance
- Thekkath and Levy — exceptions made order of magnitude faster by deferring to applications

What Can We Do?

Virtualization

Multiple guest operating systems inside a guest OS

Microkernels

Provide the minimum amount of kernel mechanisms

Exokernels

Expose hardware resources as directly as possible

Extensible Kernels

Allow guaranteed safe components into the kernel

Exokernels vs Extensible Kernels

Exokernels

- Expose underlying hardware functionality
- Kernel provides resource protection, not management
- We'll look at Aegis and ExOS

Extensible Kernels

- Provide a safe extension infrastructure
- Provide a core set of extensible services
- We'll look at SPIN

Exokernel: An Operating System Architecture for Application-Level Resource Management

Dawson R. Engler

M. Frans Kaashoek

James O'Toole

MIT Laboratory for Computer Science

The People

Dawson R. Engler

- MIT → Stanford (EE and CS)
- ACM Grace Hopper Award & SIGOPS Mark Weiser Award
- "If you know how to hack, we should meet."

M. Frans Kaashoek

- Vrije Universiteit → MIT EECS (CSAIL)
- NSF Young Investigator Award & SIGOPS Mark Weiser Award

James O'Toole Jr.

- ???

Problems and Solutions

The Problem

- Fixing OS abstractions → limited performance, flexibility and functionality
- Denies the advantages of domain-specific optimizations
- Discourages changes to existing abstractions
- Restricts the flexibility of application builders

The Solution

- Separate resource protection from management
- Allow application level management of physical resources
- Exports hardware resources through low-level interfaces
- Library operating systems implement services & policies

The "End-to-end" argument

"The exokernel architecture is founded on and motivated by a single, simple and old observation: the lower the level of a primitive, the more efficiently it can be implemented, and the more latitude it grants to implementors of higher-level abstractions".

Exokernel Design Principles

Securely expose hardware

Allow hardware resources to be accessed as directly as possible.

Expose allocation

Allow libOS to request specific physical resources

Expose names

Export physical names removing a level of indirection

Expose revocation

Well-behaved libOS can perform effective application-level resource management

Exokernel Design Techniques

Secure bindings

Allow libOS to securely bind to machine resources

Visible revocation

Allow libOS to participate in resource revocation

Abort protocol

Break secure bindings of uncooperative libOS by force

Downloading code

Downloading code into the kernel for performance and bounding

Case Studies

- Aegis: an exokernel supporting some system calls and "primitive operations"
- ExOS: A libOS implementing IPC, VM and remote comm at an application level

Aegis: System Call Interface

- `Yield` — Yield processor to named process
- `Scall` — Synchronous protected control transfer
- `Acall` — Asynchronous protected control transfer
- `Alloc` — Allocation of resources
- `Dealloc` — Deallocation of resources

Aegis: Primitive Operations

- `TLBwr` — Insert Mapping into TLB
- `FPUmod` — Enable or disable FPU
- `CIDswitch` — Install context identifier
- `TLBvadelete` — Delete virtual address from TLB

Aegis: Resource Access

- Processor time slices — CPU resources
- Exception contexts — dispatch hardware exceptions to applications
- Address translations — virtual memory
- Protected Control Translations — substrate for IPC abstractions

ExOS: Abstractions

IPC Abstractions

built atop of protected control transfer mechanisms

Application-level Virtual Memory

aliasing, sharing, caches, page allocation and DMA

Application-Specific Safe Handlers

untrusted message-handlers downloaded into kernel and verified via inspection and sandboxing

ExOS: Extensibility

- Remote Procedure Call — server managed registers
- Page table structures — inverted page tables
- Schedulers — application level stride scheduling
- Performance comparison between standard ExOS abstractions and above mentioned custom ones

Performance Evaluation

Comparisons against Ultrix and between multiple library operating systems

Looking at the Paper ...

Takeaways

- Simple primitives allow for very efficient implementation
- Low-level secure multiplexing of hardware is implemented efficiently.
- Traditional OS abstractions implemented efficiently by libraries
- Special-purpose implementations of abstractions by library modification

Extensibility, Safety and Performance in the SPIN operating system

Bershad, Savage, et al.

Department of Computer Science and Engineering, University of Washington

The People

Brian N. Bershad

- UC Berkeley → U. Washington → CMU → U. Washington
- NSF Young Investigator Award & SIGOPS Mark Weiser Award
- Appliant Inc and Illumita

Stefan Savage

- CMU → U. Washington → UC San Diego
- Asta Networks, Rendition Networks, Netsift (Cisco)

Overview

- OS that can be dynamically specialized to be safely meet the performance and functionality requirements of applications
- Motivated by the need to support applicaions whose demands are poorly matched by the OS implementation or interface

Design Techniques

Colocation

OS extensions are dynamically linked into the kernel virtual address space

Enforced modularity

Extensions cannot accesss memory or execute instruction unless explicitly given access via an interface

Logical protetion domains

Kernel namespaces containing code and exported interfaces resolved by an in-kernel linker

Dynamic call binding

Extensions execute in response to system events

Related Work

- Hydra: multi-level policies and capability based protection
- Microkernels exporting a small number of abstractions allowing application-specific extensions but with high-communication overhead
- "Little languages" that allow interpreted code in the kernel
- Software fault isolation to safely link arbitrary code into the kernel's address space

Protection Model

- Capabilities are unforgetable references to system resources, implemented by pointers checked at compile to prevent illegal access
- Protection domains define the set of accessible names available to an execution context named by a capability and used to control dynamic linking.
- Create, Resolve and Combine operations to manage domains

Extension Model

- Extensions defined in terms of services and handlers
- An event is a message that announces a change in the system state or a request for services.
- An extension install a handler for the event via a central dispatcher
- Any procedure exported by an interface is also an event.
- Primary right to handle an event is restricted to the default implementation that can arbitrate the installation of new handlers
- Guards may be used to further restrict access to events

Core Services

- Extensible memory management – physical and virtual addresses and translations
- Extensible thread management – processor contexts called strands, applications can provide their own thread model and schedulers

SPIN vs Aegis/ExOS

- SPIN uses Modula 3 features to download extensions into the kernel while protecting it.
- Aegis provides a small set of optimized primitives for application ExOS to use.
- Both systems support downloading code but this is not a core feature in Aegis/ExOS.