Extensible OSes
Exokernel and SPIN
(Lecture 9, cs262a)

Ion Stoica,
UC Berkeley
September 25, 2016

Today’s Papers

“Exokernel: An Operating System Architecture for Application-Level Resource Management”, Dawson R. Engler, M. Frans Kaashoek, and James O’Toole Jr. (https://pdos.csail.mit.edu/6.828/2008/readings/engler95exokernel.pdf)

“SPIN: An Extensible Microkernel for Application-specific Operating System Services”, Brian N. Bershad, Craig Chambers, Susan Eggers, Chris Maeda, Dylan McNamee, Przemysław Pardyak, Stefan Savage, and Emin G¨un Sirer (www.cs.cornell.edu/people/egs/papers/spin-tr94-03-03.pdf)

Traditional OS services – Management and Protection

Provides a set of abstractions

- Processes, Threads, Virtual Memory, Files, IPC
- APIs, e.g.,: POSIX

Resource Allocation and Management

Protection and Security

- Concurrent execution

Context for These Papers (1990s)

Windows was dominating the market

- Mac OS downward trend (few percents)
- Unix market highly fragmented (few percents)

OS research limited impact

- Vast majority of OSes proprietary
- “Is OS research dead?”, popular panel topic at systems conferences of the era

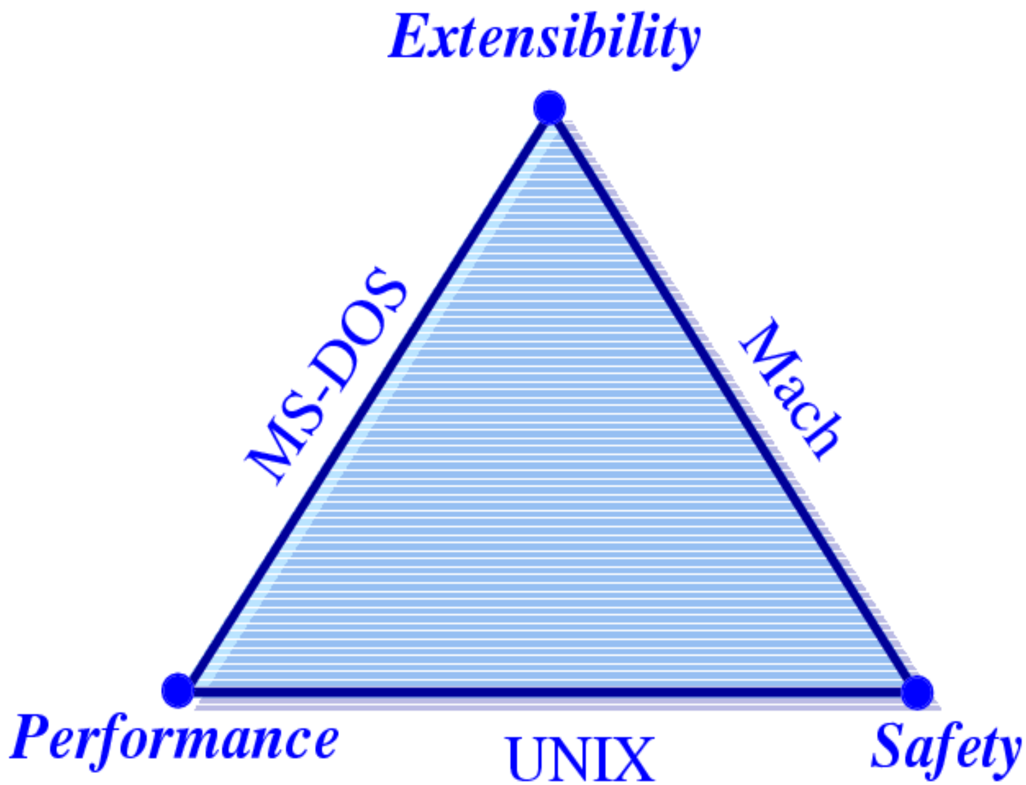
An effort to reboot the OS research, in particular, and OS architecture, in general

Challenges

Extensibility

Security

Performance



From Stefan Savage's SOSP 95 presentation

Can we have all 3 in a single OS?

Challenge: “Fixed” Interfaces

Both papers identify “fixed interfaces” provided by existing OSes as main challenge

- Fixed interfaces provide protection but hurt performance and functionality

Exokernel:

- “*Fixed high-level abstractions hurt application performance because there is no single way to abstract physical resources or to implement an abstraction that is best for all applications.*”
- “*Fixed high-level abstractions limit the functionality of applications, because they are the only available interface between applications and hardware resources*”

Challenge: "Fixed" Interfaces

Both papers identify "fixed interfaces" provided by existing OSes as main challenge

- Fixed interfaces provide protection but hurt performance and functionality

SPIN:

- "*Existing operating systems provide fixed interfaces and implementations to system services and resources. This makes them inappropriate for applications whose resource demands and usage patterns are poorly matched by the services provided.*"

Problems in existing OSes

Extensibility

- Abstractions overly general
- Apps cannot dictate management
- Implementations are fixed

Performance

- Context switching expensive
- Generalizations and hiding information affect performance

Protection and Management offered with loss in Extensibility and Performance

Symptoms

Very few of innovations making into commercial OSes

- E.g., scheduler activations, efficient IPC, new virtual memory policies, …

Applications struggling to get better performances

- The knew better how to manage resources, and the OS was “standing” in the ways

Examples Illustrating the need for App Control

Databases knows better than the OS what pages they will access

- Can prefetch pages, LRU hurts their performance

Shared virtual memory systems know whether to use disk or remote memory

- Use a page fault to retrieve page from disk / another processor

…

Two Papers, Two Approaches

Exokernel:

- Very minimalist kernel, most functionality implemented in user space
- Assumed many apps have widely different requirements

SPIN:

- Securely download functionality (code) in the kernel
- Mostly focused on protecting standard OS against device drivers

Exokernel

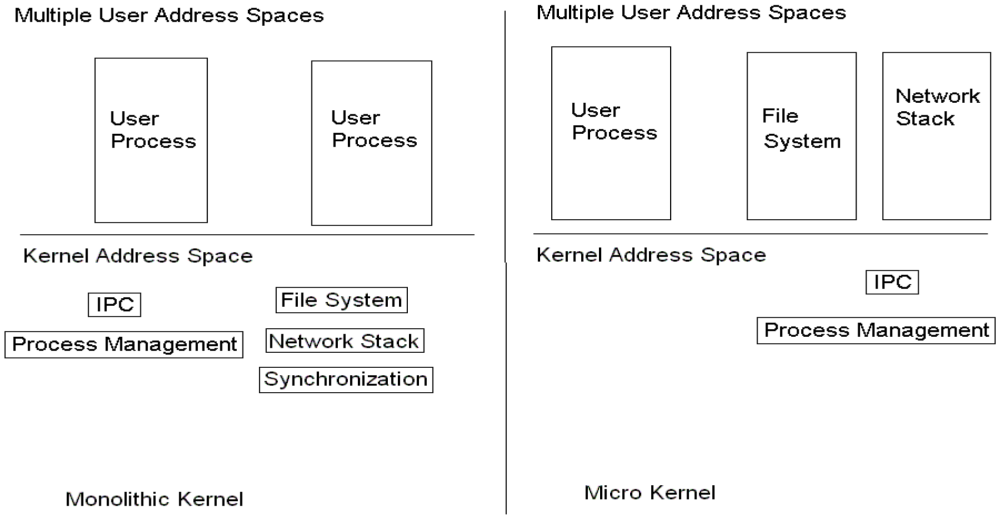
A nice illustration of the end-to-end argument:

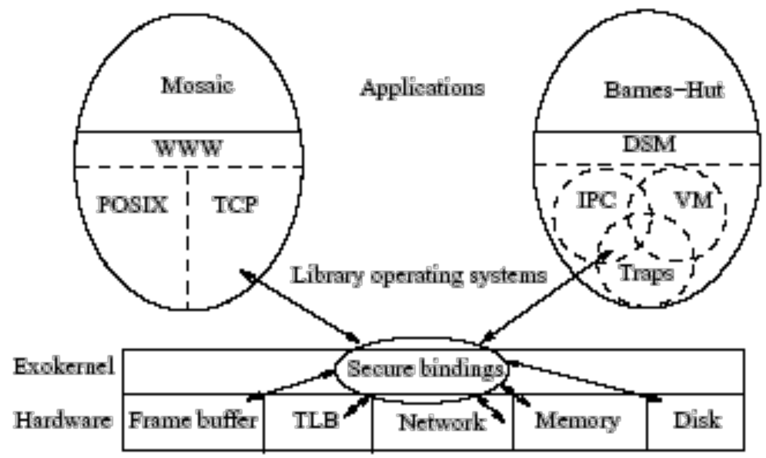
- ``general-purpose implementations of abstractions force applications that do not need a given feature to pay substantial overhead costs."
- In fact the paper is explicitly invoking it (sec 2.2)!

Corollary:

- Kernel just safely exposes resources to apps
- Apps implement everything else, e.g., interfaces/APIs, resource allocation pollcies

OS Component Layout





Exokernel

www.cs.cornell.edu/courses/CS6410/2011fa/lectures/08-extensible-kernels.pdf
(Hakim Weatherspoon, Cornell University)

Exokernel Main Ideas

Kernel: resource sharing, not policies

Library Operating System: responsible for the abstractions

- IPC
- VM
- Scheduling
- Networking

Lib OS and the Exokernel

Lib OS (untrusted) can implement traditional OS abstractions (compatibility)

Efficient (LibOS in user space)

Apps link with LibOS of their choice

Kernel allows LibOS to manage resources, protects LibOSes

Exokernel design

Securely expose hardware

- Decouple authorization from use of resources
- Authorization at bind time (i.e., granting access to resource)
- Only access checks when using resource
- E.g., LibOS loads TLB on TLB fault, and then uses it multiple times

Expose allocation

- Allow LibOSes to request specific physical resources
- Not implicit allocation; LibOS should participate in every allocation decision

Exokernel design

Expose names

- Remove one level of indirection and expose useful attributes
  - E.g., index in direct mapped caches identify physical pages conflicting
- Additionally, expose bookkeeping data structures
  - E.g., freelist, disk arm position (?), TLB entries

Expose revocation

- "Polite" and then forcibly abort
- Repossession (still leaves some small # of pages to app)

Example: Memory

Guard TLB loads and DMA

- Secure binding: using self-authenticating capabilities
  - For each page Exokernel creates a random value, check
  - Exokernel records: {Page, Read/Write Rights, MAC(check, Rights)}
- When accessing page, owner need to present capability
- Page owner can change capabilities associated and deallocate it

Self-authenticated capability

Large Software TLB (why?)

- TLB of that time small
- LibOS can manage a much bigger TLB in software

LibOS handles page faults

Example: Processor Sharing

Process time represented as linear vector of time slices

- Round robin allocation of slices

Secure binding: allocate slices to LibOSes

- Simple, powerful technique: donate time slice to a particular process
- A LibOS can donate unused time slices to its process of choice

If process takes excessive time, it is killed (revocation)

Example: Network

Downloadable filters

Application-specific Safe Handlers (ASHes)

- Preprocess packets, e.g., copy them in memory
- Can reply directly to traffic, e.g., can implement new transport protocols; dramatically reduce

Secure biding happens at download time

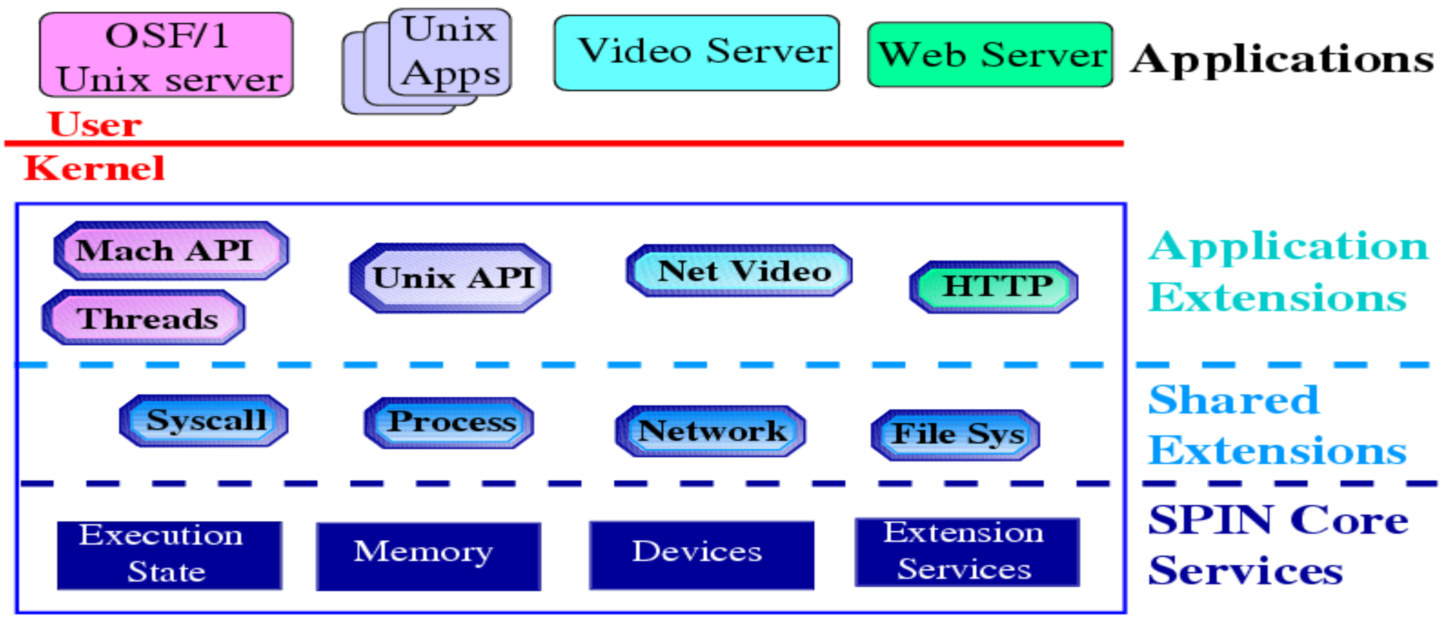
SPIN

SPIN

Use of language features for Extensions

- Extensibility
  - Dynamic linking and binding of extensions
- Safety
  - Interfaces; type safety; extensions verified by compiler
- Performance
  - Extensions not interpreted; run in kernel space

SPIN structure



From Stefan Savage's SOSP 95 presentation

SPIN Main Ideas

Extend the kernel at runtime through statically-checked extensions

System and extensions written in Modula-3

Event/handler abstraction

Language: Modula 3

Designed by DEC and Olivetti (1980s)

- Descendent from Mesa

Modern language (at that time)

- Interfaces
- Type safety
  - E.g., Array bounds checking, storage management, GC
- Threads
- Exceptions

“Died” together with DEC (acquire by Compaq in 1998)

SPIN design

Co-location

- Same memory-space as kernel

Enforces modularity

Local protection domains

- Resolves at link time

Dynamic call binding

- Event handler pattern

Events and Handler

Event: a message that announces a change in the state of the system or a request for service

(Event) Handler: a procedure that receives the message

An extension installs a handler on an event by explicitly registering the handler with the event through a central

- Essentially a callback mechanism

Example: Memory

The kernel controls allocation of physical and virtual addresses capabilities

Extensions:

- Event: page fault
- App provides handle for page faults

Example: Processor Sharing

Based on Modula-3 threads

Scheduler multiplexes processor among competing strands

- A strand is similar to a thread in traditional operating but no kernel state
- Use preemptive round-robin to schedule strands

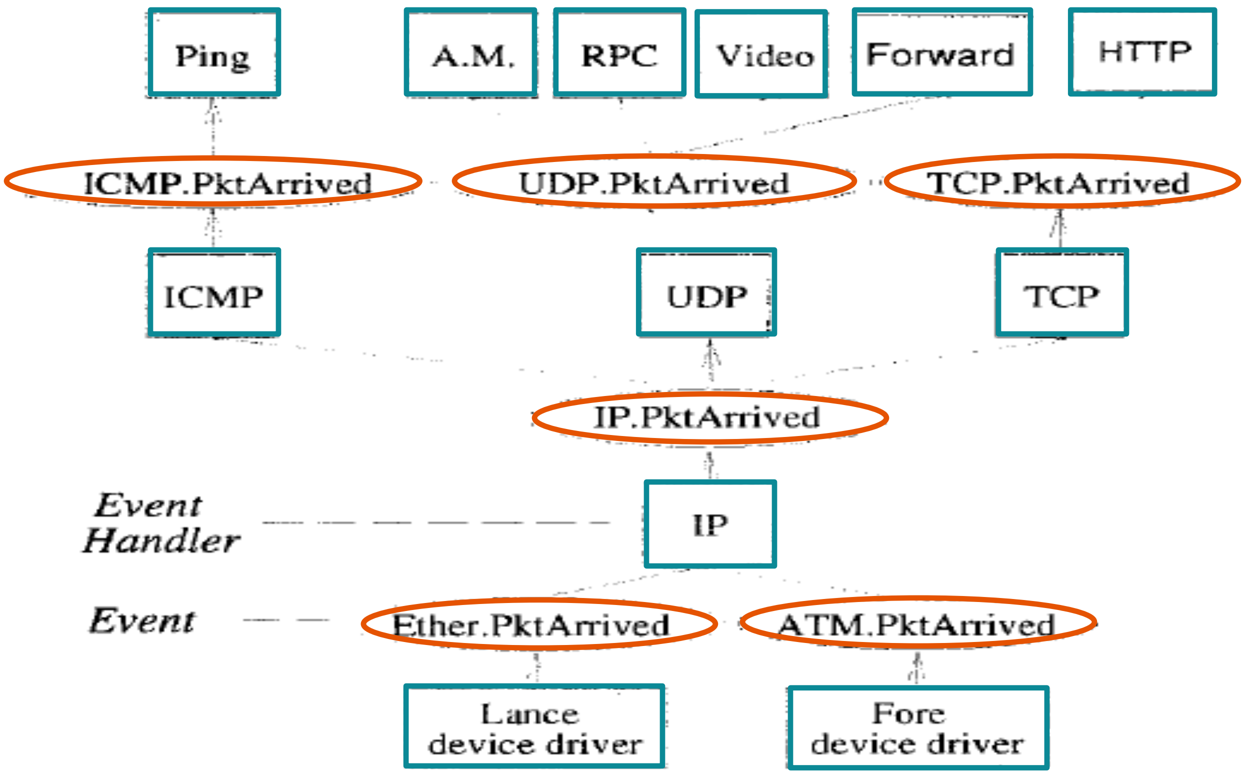
Events:

- Block (e.g., wait for IO) and Unblock (e.g., interrupt)

Handlers: need to be provided by thread package

- Checkpoint and Resume

Example: Network Stack



www.cs.cornell.edu/courses/CS6410/2011fa/lectures/08-extensible-kernels.pdf
(Hakim Weatherspoon, Cornell University)

SPIN vs Exokernel

SPIN uses programming language facilities and communicates through procedure calls

Uses hardware specific calls to protect without further specification

Summary

Extensibility without loss of security or performance

Exokernels

- Safely export machine resources
- Decouple protection from management

SPIN

- Kernel extensions (imported) safely specialize OS services
- Safety thorugh Programming Language support