Outline

- Extensible Operating Systems (contd.)
  - L3/L4, Exokernel, and SPIN
    - differences in philosophy
    - safe kernel extensions
      - rationale and mechanisms
      - protection and extension models
      - SPIN core services
  - Course Wrapup

[Bers95] Bershad et al., Extensibility, Safety and Performance in the SPIN Operating System, 15th SOSP

Microkernel and Extensible OS Philosophies

Monolithic OSes (e.g., UNIX)

- inflexible, bloated kernels
- user application
- kernel

Monolithic OSes (e.g., L3/L4, Exokernel)

- flexible, at the cost of increased IPC
- user application
- library OS
- kernel

Extensible OSes (e.g., SPIN)

- flexible, at the cost of safety/performance
- user application
- kernel

SPIN Goals

- Extensibility
  - applications must be able to extend kernel functionality
    - requires FG access to system services

- Safety
  - access to system resources must be controlled at the same granularity at which extensions are defined
    - else, this can compromise safety, or alternately performance

- Performance
  - need to support low-overhead IPC between extension and system
  - ideally, also between application and extension
    - not SPIN’s focus: examined by L3/L4 and Exokernel (protected control transfer)
**SPIN Philosophy**

- An OS can simultaneously meet goals of extensibility, safety, and performance using **language and run-time services** that provide low-cost, fine-grained, protected access to OS resources
  - language-based model of protection
  - model for integrating extensions with rest of the kernel code

- Four specific techniques
  - co-location: extensions become part of kernel virtual-address space
  - enforced modularity: language-based protection ensures low-cost isolation
  - logical protection domains: kernel namespaces
    - interactions across domains resolved by dynamic linking
    - cross-domain interactions can proceed at procedure-call speeds
  - dynamic call binding: associating extensions with system events

- Appropriately defined **system services interfaces**

**SPIN Extensions**

- Granularity of an extension: A procedure call
  - extension mechanisms place few demands on the HW
  - rely instead on language-level services such as static type-checking, dynamic linking, etc.

- SPIN extensions written in Modula-3
  - compiled using a trusted compiler and executed using a trusted run-time
  - essential features of Modula-3
    - interfaces
    - allows system services to be exposed at a fine granularity (e.g., page allocation)
    - type safety
    - a compiler can statically verify that operations invoked using a reference are in fact supported by the referent’s type
    - automatic storage management
    - ensures that a reference can serve as a capability (i.e., is unforgeable)
    - bonus features: objects, generic interfaces, threads, exceptions ...

**SPIN Protection Model**

- Protection model: Which operations can be applied to resources
  - e.g., an address-space model restricts access to own address-space

- **Problem:** Given extensions expressed in Modula-3, how to ensure that they are well-behaved (do not invoke operations that they are not supposed to)

- **Solution:** Rely on dynamic linking and Modula-3 type safety

**SPIN Protection Model: Capabilities**

- In SPIN, capabilities are just object/interface references

- Advantages
  - can be checked at compile-time: no run-time overhead
  - automatic storage management in the Modula-3 run-time ensures unforgeability

- How are capabilities passed to the application programs?
  - **Problem:** Application programs are not written in Modula-3
  - **Solution:** Externalized references
    - what gets supplied to the application is an index into a per-application table of capabilities
    - the table stores the actual capability
    - run-time check for index integrity on each request
SPIN Protection Model: Protection Domains

- Need to create protection domains (namespaces) at the language level
  - simplest choice: everything is in one module
    - too restrictive from programming viewpoint

- SPIN solution: Provide facilities for creating/coordinating/linking program-level namespaces into protection domains
  - a protection domain is a set of safe object files with exported interfaces
    - safe = signed by a trusted Modula-3 compiler, or asserted by the kernel
    - domains can be disjoint or intersecting
      - intersecting domains model sharing
    - imported symbols are left unresolved
  - RESOLVE operation: dynamic linking
    - matches imported calls in target domain to exported calls in source domain
    - once resolved, interactions can proceed at procedure-call speeds
  - COMBINE operation: collection of interfaces
    - a kernel nameserver which advertises/authorizes protection domains

SPIN Extension Model

- Extension model defines a controlled communication facility between the extensions and the base system
  - what can extensions do?
  - when/how are extensions invoked?

- What can extensions do?
  - passively monitor system activity
  - offer hints to the system, e.g., for page replacement
  - entirely replace a system service, e.g., scheduling

- When/how are extensions invoked?
  - events and handlers
  - an extension registers a handler on an event
  - kernel event dispatcher redirects handling of the event to the handler

SPIN Extension Model: Events and Handlers

- Events: Just another part of the exported interface
  - indistinguishable from a procedure

- Handlers: Procedures having the same type as the events
  - any number of handlers associated with an event
  - kernel event dispatcher specifies arguments for the handler at run-time
    - kernel can preempt handler execution
      - when only one handler, event dispatch is just a function call
      - otherwise, dynamically compiled dispatch routine

- Handler restrictions enforced by the **primary module**
  - implementation module that exports the event
  - other modules interact with the primary module
    - can deny/accept the handler
    - can associate guards for executing the handler
    - can restrict handler execution to be synchronous/asynchronous, bounded time, order of execution, etc.

SPIN Core Services

- Memory management, thread management, device drivers, etc.
  - needed in addition to extension mechanisms
  - else, nothing for extensions to access

- Memory management
  - services for physical storage, naming, translation
    - physical storage:
      - allocate(size, color/contiguity, other attributes)
        - returns a capability for the memory
      - raises an event when system needs to reclaim a page
        - application extensions can dictate how this event must be handled
    - naming:
      - request a virtual address range; again, returns a capability
    - translation:
      - installs mapping between the two capabilities (downloads into MMU)
      - raises events for protection faults, illegal address, page not present, etc.
Evaluation of SPIN Mechanisms

- System size
  - how much additional complexity does supporting extensions entail?
  - small!
    - extension support (5%), core services (20%)
    - Modula-3 support (25%)
    - low-level OS mechanisms (device drivers, etc.) (50%)

- Microbenchmarks (on 133 MHz DEC Alpha)
  - protected in-kernel call: 0.13 µs
    - interaction between kernel code and extension
  - system call: 4 µs
    - install an extension and talk with it
    - without any of the implementation tricks of L3/L4 or Aegis
  - cross address-space call: 89 µs
    - each address-space installs an extension (per RPC) that talks with the other

- Application details in the paper: Extensibility helps!

Course Wrapup

- OS Structure: Lectures 1, 2
- Process management: Lectures 3-11
  - processes, threads, IPC, scheduling
  - synchronization and deadlocks
- Storage management: Lectures 12-17
  - memory management, virtual memory
  - file systems, I/O, secondary storage
- Protection and security: Lectures 18-20
- Case studies: Lectures 21-22
  - Windows NT, UNIX/Linux
- Advanced topics: Lectures 23-26
  - encryption
  - extensible OSes

Course Wrapup (contd.)

- Final exam
  - location: 102 WWH, 1:00–3:00pm, May 11, 1999
  - particulars: 5 questions
    - OS structure, processes, IPC, CPU scheduling
    - process synchronization, deadlocks
    - memory management, virtual memory
    - file systems, I/O, secondary storage structure
    - protection, security
    - (µkernel, extensible OSes)
      - (language-based protection, encryption)

- Overdue homeworks, critiques
  - please e-mail them/put them in my mailbox in 715 Broadway
  - Project 3
    - brief (less than 5 pages) writeup about design/difficulties/final status
    - pointer to directory containing the code
      - a README file containing instructions on how to run the program
    - deadline: May 13th (final grades are due May 14th)