Extensibility, Safety and Performance in the SPIN Operating System
SPIN Architecture

SPIN is an extensible micro-kernel
- minimal kernel
- modular OS above micro-kernel
- easily tunable OS modules

SPIN uses software based protection
- micro-kernel and OS modules in same address space
- OS modules written in type safe language
- correct use of interfaces enforced statically
Extensibility

System behavior can be modified by writing extensions

- Defined in terms of events and handlers
- Low latency access to SPIN services supported via dynamic linking and colocation
- SPIN and modules written in SPIN’s safe extension language (Modular 3)
Modular 3

Support for interfaces
Type safety prevents arbitrary memory accesses
Automatic storage management
Modules communicate via procedure calls through interfaces
Safety depends on static type checking and dynamic linking
How Does It Work?

Extension code is dynamically linked into the system
- References resolved via exported interfaces

Extensions register new handlers with SPIN’s event dispatcher
- Guards (predicates) identify handlers to call

SPIN compiler ensures extensions are safe
SPIN’s Core Services

Lower-level abstractions than Mach or L4

Memory services
- physical address service
- virtual address service
- translation service

Thread management
- strands
- defines events handled by user-level and kernel schedulers
SPIN Memory Management

Physical address service
- allocation of physical page frames

Virtual address service
- allocation of virtual pages

Translation service
- manipulation of page tables and TLB
Strands

CPU contexts

Interface for user-level threads libraries
- events to signal change in thread state
- block, unblock, preempt etc

Control flow for user-level threads
Like scheduler activations
Performance Summary

Cross-domain calls are procedure calls within the same address space
OS must be written in Modular 3
Overall efficiency is better than Mach, but worse than a monolithic kernel
- but it has fine-grain protection
Some Thoughts

Is SPIN really a micro-kernel?
What is the ideal minimal kernel interface?
- what abstractions should it export?