Extensibility, Safety and Performance in the SPIN Operating System

Presented by Allen Kerr

Overview
- Background and Motivation
- Modula-3
- SPIN architecture
- Benchmarks
- Conclusion

Hardware Vs Software Protection
- Hardware
  - One-size-fits-all approach to system calls
  - Requires software abstraction
- Software
  - Applications tell the system what needs to be done
  - Allows checks to be optimized using assumptions
  - Allows untrusted user code to be safely integrated into the kernel

How Network Video works

How It Ought to Be

Motivation

Hardware
- One-size-fits-all approach to system calls
- Requires software abstraction

Software
- Applications tell the system what needs to be done
- Allows checks to be optimized using assumptions
- Allows untrusted user code to be safely integrated into the kernel

How Network Video works

How It Ought to Be

Motivation

Hardware
- One-size-fits-all approach to system calls
- Requires software abstraction

Software
- Applications tell the system what needs to be done
- Allows checks to be optimized using assumptions
- Allows untrusted user code to be safely integrated into the kernel
Modula-3

- Similar feature set to Java
  - Pointer safety
  - Exceptions
  - Interfaces
  - Modules
  - Static Type Checking
  - Dynamic Linking
- Concerns
  - Execution Speed
  - Threads, allocation, and garbage collection
  - Memory Usage
  - Mixed-Language Environment

SPIN

- Kernel programmed almost exclusively in Modula-3
- Applications can link into kernel
- Examples of services
  - Filing and buffer cache management
  - Protocol processing
  - Scheduling and thread management
  - Virtual memory

Further SPIN Motivation

- Most OSs balance generality with specialization
- General systems run many programs but run few well
- Specializing general operating system
- Costly
- Time consuming
- Error-prone

Goals

- Extensibility
  - Allow applications to extend any service
- Performance
  - Dynamically inject application code into the kernel
- Safety
  - Rely on language protection for memory safety
  - Rely on interface design for component safety

SPIN System Components

Related Work

- Hydra
  - Applications manage resources
  - High overhead
- Microkernels
  - High communication overhead
- Software Fault Isolation
  - May lack necessary flexibility
- Aegis
  - Same goals as SPIN, different implementation
**SPIN Architecture**

- Co-location
  - Low cost communication between system and extensions
- Enforced modularity
  - Extensions written in modula-3
- Logical protection domains
  - Namespaces
- Dynamic call binding
  - Calls respond to system events.

**Protection Model**

- Defines a set of accessible names
- Language level protection
  - If you have the reference, you have access
- Code is safe if signed by a modula-3 compiler
- Create
  - Creates a new domain
  - Safe object file
  - Leaves imported interface symbols unresolved
- Resolve
  - Dynamic linking
  - Resolves undefined symbols
- Combine
  - Combines 2 existing domains

**Example**

```
CREATE Domain A.  // Create domain
GET A in Domain A. // Get domain A
SERVICE A, A2.    // Service A and A2
CREATE Domain B.  // Create domain
GET B in Domain A. // Get domain B
GET B in Domain A. // Get domain B
```

**Extension Model**

- Determines the ease, transparency and efficiency of extensibility
- Communication styles
  - Passive monitoring
    - Offer hints to the system
  - Replace current functionality
    - Events
      - Announcement to the system
    - Request for service
      - Handlers
        - Procedure that receives a message
  - Right to call procedure is equivalent to right to raise an event

**Core Service**

- Kernel services that control hardware resources
  - Extensible Memory Management
  - Extensible Thread Management

**Extensible Memory Management**

- Three main interfaces
  - Physical Storage (Physical Addressing)
    - Use of pages
    - Allocation of pages
    - Controlled by core services
  - Naming (Virtual Addressing)
    - Bind to a process
    - Controlled by references
    - Translation
    - Raises exceptions related
  - Does not implement memory management directly
  - Provide base for higher levels
Memory management interfaces

EXTEND Thread Management:
TYPE T: THREAD; (fläche unique)
PROVIDES Thread Management interfaces:
  - Allowing user-defined behavior with particular constraints
  - Strand
  - Event:
    - Block
    - Unblock
  - Management only affects outside of kernel

Thread Interfaces

EXTEND Thread:
TYPE T: THREAD; (fläche unique)
PROVIDES Thread Management interfaces:
  - Allowing user-defined behavior with particular constraints
  - Strand
  - Event:
    - Block
    - Unblock

Implications for Trusted Services

- Core services interact with hardware
- They must follow their specifications
- Trust is required for extension building

System Performance

- System Size
  - Measured by lines of code and object size
- Microbenchmarks
  - Low level system services
  - Networking
    - Suite of network protocols
  - End-To-End Performance
    - Show performance of two applications

Microbenchmark results

- Shows a significant performance increase
Conclusions

- SPIN Demonstrates
  - Good performance
  - Extensibility
  - Safety
  - Ability to rely on programming language features to construct systems
  - High level programming languages can be useful in core areas of operating system design

Questions?

References

All figures used were from one of these sources
- “Extensibility, Safety and Performance in the SPIN Operating System” by Bershand
- “Protection is a Software Issue” by Bershand
- Talk titled “Language Support for Extensible Operating Systems”
- Talk titled “SPIN - An Application-Oriented Operating System”

All sources accessible through the SPIN papers website