Extensibility, Safety and Performance in the SPIN Operating System

Department of Computer Science and Engineering, University of Washington
Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers, Susan Eggers

Presented By James Whiteneck
Outline

- Overview
- Related Work
- Motivation
- Goals
- SPIN
  - Architecture (Domains and Extensions)
  - Core Services (Memory and Thread Management)
  - Performance
- Conclusion
Overview

- SPIN is an extensible OS that allows applications to alter the OS
- Executed in kernels virtual address space
- Extensions are written in a type safe language
- Dynamically linked into existing OS kernel
- Provides ability to alter core services
Related Work

- **Hydra**
  - Allows applications to manage resources through kernel (high overhead)

- **Microkernels**
  - Extendible (high overhead)

- **RPCs**
  - L3 (high overhead, 100x as much as PC)

- **Software-Based Fault Isolation**
  - Not application specific
Motivation

- Operating Systems are complex
  - Not easy to change
- Want: System that can be dynamically changed to suit specific applications
  - Safe, easy to use, good performance

Operating System Structures

- **Monolithic**
  - User Level
  - Kernel
  - Hardware

- **Microkernel**
  - User Level
  - Microkernel
  - Hardware

- **Extensible**
  - User Level
  - Kernel
  - OS
  - Hardware
SPIN

User

Kernel

Applications

Application Extensions

Shared Extensions

SPIN Core Services

OSF/1 Unix server

Unix Apps

Video Server

Web Server

Mach API

Threads

Unix API

Net Video

HTTP

Syscall

Process

Network

File Sys

Execution State

Memory

Devices

Extension Services

Goals

- **Co-location**: extensions are linked directly into kernel
  - Reduces cost of sharing data
- **Enforced modularity**: Compiler enforced boundaries
- **Protection domains**: kernel interfaces
  - Isolate failures and provide cheap context switches
  - Safely exposing kernel interfaces
- **Dynamic Call Binding**:
  - Events trigger extensions
    - Page faults, threads
SPIN - Architecture

- Software to safely combine system and application code
- Implemented in Modula-3
  - Safety
- Defined in two models
  - Protections
    - Controls access to resources through capabilities
  - Extensions
    - Defined in terms of events and handlers
Protection Domains

- Capabilities: is a reference (pointer) to a system object, interface, or collection of interfaces
  - Protection provided by compiler
  - Can’t be changed
- Protection is at the language level, not in virtual memory
Extensions

- Extension model provides controlled communication between extensions (events and event handlers) and system

- Event
  - Message that announces a change in system state or requests a service

- Event handler
  - Procedure that receives messages from events

- Events sent through central dispatcher
SPIN – Core Services

- Memory Management
  - Three service interfaces
    - Physical Address
      - Physical page allocation and use
    - Virtual Address
      - Capability allocation
    - Translation
      - Mappings between physical and virtual addresses
SPIN – Core Services

- Thread Management
  - Strands
    - Similar to threads
    - No state in kernel
  - Set of events and event handlers
  - Interfaces provide scheduling, concurrency, synchronization
    - Application specific
Trust

- SPIN trusts core services
  - Required to mediate access to resources, applications, and extensions
- Extension failures only affect their own extension
SPIN - Performance

- **OS Model Comparisons**
  - SPIN v0.4
  - DEC OSF/1 v2.1 (monolithic)
  - MACH v3.0 (microkernel)

- **Compared**
  - System Size
  - Microbenchmarks
  - Networking
  - End-to-end Performance
Performance - Microbenchmarks

Protected Communication

<table>
<thead>
<tr>
<th>Operation</th>
<th>DEC OSF/1</th>
<th>Mach</th>
<th>SPIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected in-kernel call</td>
<td>n/a</td>
<td>n/a</td>
<td>0.13</td>
</tr>
<tr>
<td>System call</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Cross-address space call</td>
<td>845</td>
<td>104</td>
<td>89</td>
</tr>
</tbody>
</table>
# Performance - Microbenchmarks

## Thread Management

<table>
<thead>
<tr>
<th>Operation</th>
<th>DEC OSF/1</th>
<th>Mach</th>
<th>SPIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kernel</td>
<td>user</td>
<td>kernel</td>
</tr>
<tr>
<td>Fork-Join</td>
<td>198</td>
<td>1230</td>
<td>101</td>
</tr>
<tr>
<td>Ping-Pong</td>
<td>21</td>
<td>264</td>
<td>71</td>
</tr>
</tbody>
</table>
Performance - Microbenchmarks

Virtual Memory

![Bar chart showing performance metrics for different operations and systems: DEC OSF/1, Mach, SPIN. The chart includes operations such as Dirty, Fault, Trap, Prot, Prot100, Unprot100, Appel1, Appel2, and their corresponding microsecond values for each system.](chart.png)
Other Tests

- Networking
  - Measured round trip latency and bandwidth
  - SPIN application code executes at kernel level
    - Low latency access to both device and data

- End-to-end
  - Measured number of clients serviced by a networked video server
    - SPIN allowed same number of clients for less CPU
Conclusion

- An extensible OS can achieve good performance without compromising safety.
- Able to provide a customizable system using a base set of core services to build upon.
- Future OS should take advantage of compiler safety and support for programming languages.