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 George Dittmar
Overview

- SPIN is an extensible OS which can be dynamically specialized to safely meet the requirements of applications.
  - Has an extension infrastructure built on Module-3 interfaces.
  - Allows applications to safely change kernel services for better performance.
Motivation

- Current OS implementations forced to balance general design view or a specialized design view.
- An extensible system is one that can change to the needs of a specific application.
  - Allows for a general OS to suddenly become specialized within an applications domain.
Goals of SPIN

- Make a generalized extensible OS that is safe and has good performance.
  - Extensibility: interfaces to system services that an application can modify.
  - Safety: require application access to be controlled at same granularity of the extension definition.
How do they achieve the goals?

- SPIN relies on four techniques implemented at language level / runtime.
  - Co-location: OS extensions dynamically linked at runtime into the kernel virtual address space.
  - Enforced modularity: Extensions written in Modula-3 with the compiler enforcing interface boundaries.
  - Protection domains: kernel name-spaces that contain the exported interfaces.
  - Dynamic call binding: extension execution in response to events.
Why do we need this?

- System specialization can be costly and error prone.
- Extensible systems can be made to change based on the need of the application.
Architecture

- SPIN has software infrastructure to merge system and application functionality.
  - Implemented in Modula-3.
  - Extensions defined at granularity of a procedure call by use of an Extension Model.
  - Protection model controls the operations on a resource.
Modula-3

- SPIN leverages the built in functionality of the language to help with safety and protection of the system and extensions.
  - Support for interfaces
  - Automatic storage management
  - Generics
  - Objects
  - Etc...
Protection Model

- Control a set of operations for resources.
  - For example protection model for memory address space.
- Capabilities are unforgeable references to resources.
  - Implemented as pointers.
  - Kernel resources are all capabilities.
  - Ensures that extensions can only access resources they have been given.
- Protection domain defines a set of names that are available for an extension.
  - Has a set of capabilities that corresponds to one or more object files with one or more exported interfaces.
Protection Domain

- Defines a set of names or symbols that are referenced by code.
- Operations on a domain.
  - Create: starts a domain.
  - Resolve: basis for dynamic linking.
  - Combine: union of existing domains.
Extension Model

- Extensions allow for modification of system services.
- The model determines how easy and transparent an extension can be applied.
- SPIN has a controlled communication interface between the extensions and base SPIN kernel.
- Extensions are defined with events and handlers.
  - Event: procedure exported from the interface.
  - Handler: procedures having the same type.
Extensible Memory Management

- SPIN does not force an address space model but allows for implementation of many different models.
  - Ex. Applications with Unix address semantics.
- Broken down into three service interfaces:
  - Physical Address: use and allocation of physical pages.
  - Virtual Address: allocates capabilities.
  - Translation: mappings between the above two.
Extensible Threads

- SPIN defines the structure on which the thread model rests, not the model itself.
  - Defined as a set of events that get raised and handled by schedulers.
- Interfaces for scheduling, concurrency, and synchronization of threads.
  - Applications can include their own threads and schedulers!
  - Does not modify control flow for kernel.
Trust issues

- SPIN brings up the issue of trust with core system services.
- Make sure things are being done within interface specifications.
  - Core services are trusted by SPIN.
  - An extensions failure stays within that extension!
Evaluation

- Comparison of three OS models. All on the same hardware platform.
  - SPIN
  - DEC OSF/1 V2.1 (Monolithic Kernel)
  - Mach 3.0 (Micro Kernel)
Evaluation

- System Size
  - Size in terms of lines of code for components.
- Micro-benchmarks
  - Overhead for low level system services.
- Networking
  - Extension services allow for integration of high-performance protocols.
- End-to-End performance
  - How applications benefit from extensibility.
Micro-benchmarks

- 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>.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>

Table 2: Protected communication overhead in microseconds. Neither DEC OSF/1 nor Mach support protected in-kernel communication.

- Virtual Memory

<table>
<thead>
<tr>
<th>Operation</th>
<th>DEC OSF/1</th>
<th>Mach</th>
<th>SPIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dirty</td>
<td>n/a</td>
<td>n/a</td>
<td>2</td>
</tr>
<tr>
<td>Fault</td>
<td>329</td>
<td>415</td>
<td>29</td>
</tr>
<tr>
<td>Trap</td>
<td>260</td>
<td>185</td>
<td>7</td>
</tr>
<tr>
<td>Prot1</td>
<td>45</td>
<td>106</td>
<td>16</td>
</tr>
<tr>
<td>Prot100</td>
<td>1041</td>
<td>1792</td>
<td>213</td>
</tr>
<tr>
<td>Unprot100</td>
<td>1016</td>
<td>302</td>
<td>214</td>
</tr>
<tr>
<td>Appel1</td>
<td>382</td>
<td>819</td>
<td>39</td>
</tr>
<tr>
<td>Appel2</td>
<td>351</td>
<td>608</td>
<td>29</td>
</tr>
</tbody>
</table>

Table 4: Virtual memory operation overheads in microseconds. Neither DEC OSF/1 nor Mach provide an interface for querying the internal state of a page frame.
Micro-benchmarks cont.

- 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></td>
<td>DEC</td>
<td></td>
<td>Mach</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fork-Join</td>
<td>198</td>
<td>21</td>
<td>101</td>
</tr>
<tr>
<td>Ping-Pong</td>
<td>1230</td>
<td>264</td>
<td>338</td>
</tr>
</tbody>
</table>

Table 3: Thread management overhead in microseconds.
Networking

- Latency and bandwidth

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DEC OSF/1</td>
<td>SPIN</td>
</tr>
<tr>
<td>Ethernet</td>
<td>789</td>
<td>565</td>
</tr>
<tr>
<td>ATM</td>
<td>631</td>
<td>421</td>
</tr>
</tbody>
</table>

Table 5: Network protocol latency in microseconds and receive bandwidth in Mb/sec. We measure latency using small packets (16 bytes), and bandwidth using large packets (1500 for Ethernet and 8132 for ATM).

- Protocol forwarding

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DEC OSF/1</td>
<td>SPIN</td>
</tr>
<tr>
<td>Ethernet</td>
<td>2080</td>
<td>1420</td>
</tr>
<tr>
<td>ATM</td>
<td>1730</td>
<td>1067</td>
</tr>
</tbody>
</table>

Table 6: Round trip latency in microseconds to route 16 byte packets through a protocol forwarder.
End-to-End

- Built a network video system for testing
- Found that SPIN’s implementation allowed for the same number of clients to be serviced as DEC but used LESS CPU resources.

![Graph showing CPU utilization vs number of clients](image)

Figure 6: Server utilization as a function of the number of client video streams. Each stream requires approximately 3 Mb/sec.
Conclusions

- SPIN can achieve good performance without compromising safety.
- Using languages that have certain features allows developers to leverage compilers and runtime services to create systems in which structure and performance are complimentary.