Extensibility, Safety, and Performance in the Spin Operating System

- Brian Bershad, Steven Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc Fiuczynski, David Becker, Craig Chambers, and Susan Eggers

- Department of Computer Science and Engineering, University of Washington

- Presented by Michael Reames
SPIN

- “A flexible operating system that helps applications run fast but doesn't crash”
- OS that can be dynamically specialized to safely meet the performance and functionality of an application.
- general purpose OS that is extensible, safe, and efficient
- built around the concept of extensions and events
Extensions

- an extension changes the way the system provides services
- defined in terms of events and handlers
- only the code that needs low-latency access to system services needs to be written in system's safe extension language (Modula-3)
- dynamically linked to the kernel (co-location)
SPIN and extensions are written in Modula-3 depends on language's safety and encapsulation mechanisms specifically:

- support for interfaces
- type safety to prevent code from accessing memory arbitrarly (enforced at compile time)
- automatic storage management
- modules communicate via procedure calls through interfaces
Goals of SPIN

- Provide applications:
  - an infrastructure to allow fine grained access to system services
  - protection from the actions of others
  - low overhead for communication between applications, their extensions, and the system

- Provide system:
  - software infrastructure for combining system and application code
Motivation

- OS is either highly specialized and works really well for one type of application
- or is general and works OK for many applications
- saw extensibility in application programmers directly modifying weakly protected OS's (DOS, Windows, MacOS) and wanted to do it safely
SPIN Architecture

- makes few demands on the hardware but relies heavily on the language level services (static type checking and dynamic linking)
- any procedure call in the system is an event
- any procedure running in the system is a handler
- can change the system by changing the handler
- allows extensions to add a module that can communicate within the system at procedure call level
What we will discuss

- Implementation
  - extension model – how it works
- Safety
  - protection model – why it is safe
- Performance
  - Is it worth it?
Extension Model

- specifies how extensions work
- determines how easy it is to implement extensions
- uses extensions to help your application in a variety of ways
  - monitor system, provide information to help guide decisions, add additional handler, or replace default handler completely
Extensions

- as stated – defined in terms of events and handlers
- incorporated into the system in 2 steps:
  - extension code is dynamically linked into the kernel and unresolved references in extension are resolved by the system's exported interfaces (allows the extension to call into the system)
  - extension registers handlers with the dispatcher (allows the system to execute extension code)
Handlers

- procedures
- can have multiple handlers for an event
- guards, procedural predicates, are added to determine which handlers to invoke (can be one or many)
- can determine how events are handled
  - synchronously or asynchronously, in a particular order, with restraints on time
  - maybe only when event is raised by handler's application
Protection Model:

- safely allows user code to execute in the kernel
- protection of individual kernel resources based on capabilities, which are unforgeable references to a resource
  - extensions only access resources they have been given permission to
- instead of hardware, SPIN implements capabilities with pointers that are supported by the language
- Modula-3 compiler checks for pointers being forged or dereferenced inconsistent with their type
Protection Continued

- only extensions cleared by the compiler are allowed to be linked into the kernel (deemed safe)
- when registering handlers the dispatcher sends the new handler to the event authority to verify
  - ensure it has proper type, arguments, return values
  - ensure guard has the same and has return value of a boolean
- will only allow safe handlers to be added
- event authority can limit time a handler can run to prevent run away handler
Core Services

- the protection and extension models provide the framework for managing interfaces between applications and modules within the kernel

- **but** applications are mostly concerned with changing the way the processor and memory work

- SPIN provides a set of core services for managing memory and processors
Exported Interfaces

- the core services export interfaces with fine-grained operations
- provide simple functionality over a small set of objects
- can allocate a single virtual page, a physical page, and create the mapping between them at the cost of a procedure call
- provides cheap control over system events
Extensible Memory Management

- allows fine grained control over physical and virtual memory resources
- is efficient and safe due to SPIN's low-overhead communication between extensions and the system and its protection model
- breaks memory system into 3 basic units and provides an interface for each
  - physical address services
  - virtual address services
  - translation services
3 Memory Services

- **physical** – the use and allocation of physical pages
  - cannot be addressed directly by extensions or user applications, must receive a capability
- **virtual** – allocates capabilities for virtual addresses
- **translation** – interprets references between the two and provides mappings
- easily allow application to tailor memory management
Extensible Thread Management

- allows customizable thread package and scheduler that execute in the kernel
- combine the execution model, synchronization, and scheduling in unique way and have it operate close to the kernel and services
- doesn't define a model but the structure for one
  - defined by a set of events that are raised or handled by schedulers and thread packages
Strands

- processor contexts (like threads)
- application specific thread package defines the strand interface for its threads
- thread package and the scheduler implement the control flow mechanism for its user level contexts
- extensions only control the scheduling of threads in the user level, kernel is still responsible for kernel threads
Thread Events

- raised to signal change in strand's state
- block – driver can tell scheduler to block a strand while I/O completes
- unblock – interrupt handler can unblock the strand when I/O complete
- block and unblock events allow scheduler to communicate with thread package for saving and restoring execution state
- user level thread management in the kernel
SPIN components

- SPIN can be used as a stand alone OS
- has 5 components:
  - sys – extensibility, naming, domains
  - core – VM, scheduling, file system, device mgmt.
  - rt – runtime system
  - lib – Mod-3 libraries and handles, data structures
  - sal – low level device drivers and MMU
Performance of SPIN

- using null procedure call, in microseconds
- in kernel is procedure call between linked domains
- system call crosses user-kernel boundary, executes, and returns
- cross address space call is application calling into kernel(sys call), to another address space in kernel (proc call), and back out.

<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 call</td>
<td>845</td>
<td>104</td>
<td>89</td>
</tr>
</tbody>
</table>
Performance cont.

- thread management:
  - fork-join: time to create, schedule, and terminate new thread
  - ping-pong: time for 2 threads to synchronize
  - layered – user level library layered on kernel extensions (use Mach's kernel thread interface)
  - integrated – kernel extension that exports C-threads interface, strands, and integrated with kernel

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DEC OSF/1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Spin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kernel</td>
<td>user</td>
<td>kernel</td>
<td>user</td>
<td>kernel</td>
<td>user</td>
</tr>
<tr>
<td></td>
<td>layered</td>
<td>integrated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fork-join</td>
<td>198</td>
<td>1230</td>
<td>101</td>
<td>338</td>
<td>22</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111</td>
</tr>
<tr>
<td>ping-pong</td>
<td>21</td>
<td>264</td>
<td>71</td>
<td>115</td>
<td>17</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>85</td>
</tr>
</tbody>
</table>
Performance cont.

- virtual memory management:
  - dirty – time to query the status of a VM page
  - fault – time to notify app of page fault, enable access to the page, and resume
  - trap – latency between page fault and when handler executes

<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>
Conclusion

- allow user code in the kernel
- modify behavior of the system at procedure call level within the kernel and per application
- language support for safety
- thoughts, ideas, or concerns?
- if time...examples of using SPIN
Networking

- implemented set of network protocol stacks with SPIN
- SPIN allows user code to be dynamically linked into the stack
- incoming packets are “pushed” up the stack by events, while the handlers “pull” the packet up
- in traditional case any application code is run in the user level where each packet send involves a trap and several copies
- SPIN has it execute in the kernel with extensions
Networking cont.

- handling packets entirely in the kernel reduces latency, but throughput is the same for either system

- an example with protocol forwarding to load balance service across many servers handled within the kernel to increase efficiency
Others

- the website server for SPIN operates completed in the kernel

- networked video system with server and client:
  - server has 3 extensions – local file system to read video frames from the disk, one to send files across the network, and third as a handler for SendPacket event
  - client – extension awaits incoming packets, decompresses and then writes directly to the frame buffer
  - no need to copy the data as crosses into kernel