Extensibility, safety, and performance in the SPIN operating system

by B. Bershad, et al.

presented by M. Hoffman
Goals of the SPIN kernel

- microkernels are slow because of context-switching
- instead, use a kernel extension mechanism
  - dynamically linked
  - executes in the kernel’s address space
  - type-safe language for extensions
Whither L3?

- L3 paper at SIGOPS ‘93; This is from ‘95
- L3 on 50mhz 486; SPIN on 133mhz Alpha
- Related work doesn’t mention L3 at all.
- “a protected procedure call into server code in 12 µsecs.” (p.2)
- L3 could perform IPC in 5–7 µsecs
The SPIN architecture
Capabilities

- Just a type-safe reference in memory to a variable or procedure
- Extensions in the kernel must be written in Modula-3, so type-safe (or explicitly trusted)
- Extensions have no way to access pointers into arbitrary regions of memory
- Crossing the kernel/user boundary must translate back into type-safe references
Domains

- Sort of like a “package/namespace” in your favorite language
  - Domains named by capabilities
  - specify read/write/modify rights, etc.
- Create a domain from a signed object-file
- Resolve two domains
  - replaces unresolved symbols in target, and only those, with symbols in source
• “extensions and the kernel execute in the same virtual address space...”

• ... how can the SPIN guarantee that poorly coded extensions won’t crash the OS ...?

• compare extensions in the extensible operation system with Linux.

• Linux kernel modules have no such guarantees
- ... how would the architecture handle being programmed in another language?
- probably not well depending on the language
- Are these safety features actually part of the architecture of the system or just the language?
- the language, but they go hand in hand
• how does the system ensure the extension is valid after runtime?

• If one gains access to a Modula-3 compiler/src, would this not present a security hole?
• Is it now necessary for every system that is implemented for SPIN to be separated into two domains (public and private)?
  • not necessary. a good idea, probably.

• Also, how are domain (namespace?) conflicts handled?
  • Domains are not quite namespaces
  • More of a tree
Events

- access controlled via domains
- also have guards, predicates defining when handler is called
- only one handler can return a value to the raiser
- sounds like they can only be raised from within kernel space
Core services

- core services provide interfaces to hardware
- memory management
- processor management
- statically linked in, but can be overridden
- again, like micro-kernel “servers” but imbedded within the kernel
- requires return of resources on request
Process management details

- scheduler multiplexes *strands* via events
- *checkpoint* and *resume*
- applications can also implement their own scheduler
- can’t overload scheduler or kernel threads
- seems like a continuation of the “scheduler activations” idea
• the SPIN system is basically based on event-based paradigm... was that choice advantageous for extensibility and safety?

• events are really only used to provide hooks inside the kernel

• other than that its “thread based”
System calls

- Externally a standard system call interface
- ... but in the kernel just an event
- as a result new system calls can be defined
- just have to create a handler
Results
Page faults

- micro-benchmark on handling a page fault
- slow because of 5 required kernel/userspace crossings

<table>
<thead>
<tr>
<th>Mach 3.0 kernel</th>
<th>DEC OSF/1 kernel</th>
<th>SPIN extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>300</td>
<td>17</td>
</tr>
</tbody>
</table>

- Main speed increases a result of everything being in kernelspace
## Thread management

<table>
<thead>
<tr>
<th>Kernel Thread Operation</th>
<th>Mach 3.0 kernel</th>
<th>DEC OSF/1 kernel</th>
<th>SPIN extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>41</td>
<td>332</td>
<td>5</td>
</tr>
<tr>
<td>Ping-Pong</td>
<td>71</td>
<td>21</td>
<td>29</td>
</tr>
<tr>
<td>Terminate</td>
<td>18</td>
<td>260</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Thread Operation</th>
<th>Mach 3.0</th>
<th>DEC OSF/1</th>
<th>SPIN layered</th>
<th>SPIN native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fork</td>
<td>50</td>
<td>1131</td>
<td>103</td>
<td>20</td>
</tr>
<tr>
<td>Fork, Run</td>
<td>233</td>
<td>1164</td>
<td>157</td>
<td>64</td>
</tr>
<tr>
<td>Ping-Pong</td>
<td>115</td>
<td>233</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>ForkJoin</td>
<td>338</td>
<td>1026</td>
<td>223</td>
<td>110</td>
</tr>
</tbody>
</table>
System call overhead

• They say the DEC OSF/1 server is closest ability-wise to theirs

<table>
<thead>
<tr>
<th>Xfer Bytes</th>
<th>Mach 3.0 kernel</th>
<th>DEC OSF/1 server</th>
<th>DEC OSF/1 kernel</th>
<th>SPIN extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>309</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>128</td>
<td>9</td>
<td>453</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>
Cross-address space RPC

- OSF/1 uses ports and Sun RPC
- SPIN uses syscalls
Various questions
Their video example

- Should they have done more testing of the operating system as a whole against other operating systems? [other than the video example]
  - Answer: yes, probably.
  - Also, why didn’t they test against Mach?
• Is SPIN a failure since it is not used in any commercial or public domain OS?
• This paper introduces an “in kernel file system” (p5) SFS. Are most file systems in kernel or not?

• How is SFS different from other file systems?
• Why would fault isolation not be appropriate for systems that with fine-grained sharing?

• maybe because their event-system is more flexible?

• you would then need a good pseudo-IPC system