SPIN Operating System

• **Motivation**: general purpose, UNIX-based operating systems can perform poorly when the applications have resource usage patterns poorly handled by kernel code

Why? Current crop of monolithic or microkernel based OSes make most important decisions for us: VM policies, scheduling, etc... Extensibility is generally poor or comes with dangers/limitations (Linux modules for example)

• **Objective**: a general purpose OS capable of handling specialized needs as well, with better performance and no "new" security threats:

- **Fine-grained** access to system services, via **extensions**
- **Co-location**: extensions live in kernel space (as for Linux modules, but with some differences)
- **Logical Protection Domains**: namespaces inside kernel, intra-domain communication possible at cost of procedure call
- **Dynamic call binding**: system events are dispatched to extensions

• **Approaches to extensibility** can vary among existing/proposed solutions:
  - **Linux Modules**: very powerful, but we cannot allow any user/application to install a module at runtime
  - **Interpreted languages**: limited functionality, slow
  - **Software fault isolation**: maybe good for protection, but does not define extension interface
  - **Language-level protection**: SPIN uses simple pointers, and lets Modula-3 guarantee protection

• **Modula-3**: relevant features that make extensions provided by applications safe to run in kernel space
  - **Pointers cannot be forged**: therefore an extension can only access the symbols exported through the public interfaces of any module (i.e. *SpinPublic*)
  - **Pointers can only be de-referenced in a type-safe way** (this check is also ensured across procedure calls between different domains)
  - **Protection Domains** (vs traditional VM approaches); Create, Resolve (dynamic linking), Combine (multiple domains)

• **Events & Handlers**: basis of extension model
    - An **event** is associated with change in system
    - Extensions install **handlers**, procedures invoked by an internal dispatcher when routing events
    - **Primary implementation module** (itself a handler) decides whether to admit or reject the installation of a new handler; it also has control over the execution (synchronous/asynchronous) of a handler

• **Memory Management**: three components
    - **physical address service** manages allocation of physical pages
    - **virtual address service** manages capabilities for virtual addresses
    - **translation service** manages relationship between physical and virtual addresses
    - extensions use the events raised by these three modules to create their own memory management schemes

• **Thread Management**: each application provides its own package (execution model + constructs) and scheduler; SPIN only specifies primitive execution contexts: **strands**. Note: applications only schedule application threads execution time, **not** kernel

• **Protected communication** is possible between kernel and extensions, and between applications and services installed into the system at the cost of a procedure call.

• **Networking**: by running application inside kernel (as an extension) there is no overhead in packet processing – i.e. no context switch is necessary.

- **Latency** is greatly improved
- **Packet forwarders** are fast and can be installed by applications
- **Caching for servers**: normally we have two options, avoid caching and rely on underlying fs, or implement a high-level caching scheme and risk double-buffering. With SPIN a server can execute in kernel space and implement its own caching scheme without double-buffering

• **Conclusions**: it is possible and desirable to rely on a programming language with appropriate compile- time and run-time features to extend the functionality of an operating systems.

What are these features? The authors suggest that a minimal set:

- **Interfaces**, to declare the visible parts of a module
- **Type safety**, to prevent pointer forging, array bounds checking
- **Automatic storage management**, to prevent deallocated memory from being reused for a different object type

The results favor SPIN heavily, despite the fact that these language features guaranteed by Modula-3 may be considered to be unwanted overheads (esp. array bounds checking)