L02b: The SPIN Approach

Introduction:

- The SPIN and ExoKernel approaches aim to achieve extensibility of the OS without compromising protection or performance. Both are trying to solve the following issues:
  - µKernel approach compromises performance due to frequent border crossings.
  - Monolithic approach is not extensible.
- What do we need in an OS structure?
  - A thin OS (like µKernel): Only mechanism (how things can be done) should be ingrained in the kernel, not policies (what things can be done).
  - Access to resources without (or with minimal) border crossings (like MS-DOS).
  - Flexibility for resource management (like µKernel) without sacrificing protection and performance (like Monolithic OS).

Approaches to Extensibility:

- Hydra OS (Capability-based):
  - Kernel **mechanisms** for resource allocation.
  - Capability-based resource access: Capabilities are essentially like keys that are presented to the OS during access requests to ensure resource security. Each time a capability is passed from one object to another, its validity must be checked (which introduces overhead).
  - Resource managers are implemented as coarse-grained objects (large components) to reduce border crossings (also reduces extensibility).
- Mach OS (µKernel-based):
  - Provides very limited mechanisms in the µKernel.
  - Implements all the OS services as normal processes that run above the kernel (extensible + portable).
  - Low performance was a disadvantage in this structure.
- SPIN approach:
  - Co-location a minimal kernel and its extensions in the same space to avoid border crossings.
  - Compiler-enforced modularity to maintain protection (strongly-typed programming language).
  - Logical protection domains: Data structures provided by the Modula-3 programming language to serve as containers for the OS extensions, which free us from relying on the HW address spaces for protection.
  - Dynamic call binding: Extensions execute in response to system events. Events are declared within interfaces and can be dispatched with the overhead of a procedure call.

Modula-3 Safety:

- Strongly-typed language with built-in type safety and auto storage management.
- Supports objects (known entry points - unknown implementation), threads (executes in the context of an object), exception and generic interfaces (to expose the externally-visible methods inside objects).
- Modula-3 safety features allows implementing system services as objects with well-defined entry points.
- What you can do from outside the object is only what the entry point methods allow you to do. Which means we get the safety advantages of a Monolithic kernel without having to put the system services in a separate HW address space. This facilitates both protection and performance.
- The generic interfaces allow for creating multiple instances of the same service.
- Fine-grained protection via capabilities:
    - A HW resource can be an object (e.g. page frame).
    - An interface can be an object (e.g. page allocation module).
    - A collection of interfaces can be an object (e.g. virtual memory).
- Capabilities here are not similar to Hydra OS. In Modula-3, pointers serve as capabilities to the objects.

SPIN Mechanisms for Protection Domains:

- Create:
    - Creating a logical protection domain.
    - This mechanism allows initiating the object file and exports the names that are contained as entry point methods inside the object.
- Resolve:
    - Resolves the names that are implemented in one object file (source) and used in another one (target).
    - Similar to linking in any compiler.
- Combine:
    - Combining two different objects to create an aggregate domain.
- These mechanisms facilitate extensibility through creating and managing different domains of the same services customized for each application and running concurrently on the same HW.

SPIN Mechanisms for Events:

- External events include:
  - External interrupts.
  - Exceptions.
  - System calls.
- SPIN uses an event-based communication model.
- Services can implement event handlers that are managed by the SPIN Event Dispatcher.
- SPIN supports different mapping mechanisms between events and handlers:
  - One-to-one.
  - One-to-many.
  - Many-to-one.

Default Core Services in SPIN:

- Any OS should provide core OS services (CPU scheduling, memory management, etc.). However, an extensible OS, like SPIN, should not dictate how these services should be implemented.
- SPIN provides interface procedures for implementing these services.
- Memory management:
  The following interfaces are provided by SPIN as header functions (interfaces) to the extension implementer, who in turn should write the actual code for these header functions:
  - Physical address interfaces.
  - Virtual address interfaces.
  - Translation: Create/destroy address space, add/remove mapping.
  - Event handlers: Page fault, access fault, bad address.
- CPU scheduling:
  Same as memory management, the extension implementer should write the actual code for these header functions:
  - SPIN abstraction: Semantics defined by the extension that is running on top of SPIN.
  - Event handlers: Block, unblock, checkpoint, resume.
  - SPIN global scheduler: Interacts with the different extension threads package that are concurrently running on top of SPIN.
- Core services are trusted services.
- Extension to core services affects only the applications that use that extension. That means if something goes wrong with a particular extension, other extensions will not be affected.