Extensibility, Safety and Performance in the SPIN OS

SOSP ’95

(Also including other related papers)
Preface

• An extensible OS is desirable to balance generality and specialization, and a successful extensible system requires extensibility, safety and performance;

• From software engineering’s perspective, it is difficult to achieve non-functional properties. It may be more difficult to build an operating system that has non-functional properties;

• However, SPIN seems to possess the above three properties in a natural way. Let’s have a deep examination of SPIN and its construction …
Outline of the talk

• Background
  – Software architecture
  – Modula-3
• SPIN architecture
  – Protection model
  – Extension model
• Core services
  – Memory management
  – Thread management
• System performance  [ If time permits]
• Summary
Background

• Software architecture
  – Structures and behavior
    Structures determine the non-functional properties
  – Components and connectors
    Connectors determine the structures
  – Examples of connectors
    Procedure call (Explicit call)
    Event-driven mechanism (Implicit call)
Background

- Software architecture
  - Implicit call V.S. Explicit call
    Knowledge of targets;
    Support for reuse and evolution;
    Control transfer and data exchange;
    Reasoning about correctness

Observation: These two kinds of connectors are complementary to each other.
• Modula-3
  – What is Modula-3?
    1) Interfaces and Modulas
    2) Type safety
    3) Garbage collection
  – Why Modula-3?
    1) Why not use an ADL (Architectural Description Language)?
    2) Why not C?
    3) Why Modula-3?
Background

• Modula-3
  – How to use Modula-3? (I have answers :-)
    Sub-question 1: Which parts of SPIN are written in Modula-3? Extensions or the kernel? How about the application code?
    Sub-question 2: How does SPIN add implicit call to Modula-3?
    Sub-question 3: What modifications are performed on Modula-3 and it’s runtime?
    Sub-question 4: How does SPIN rely on Modula-3’s static type checking and dynamic linking?
SPIN Architecture

• Four techniques at language level
  – Co-location
  – Enforced modularity
  – Logical protection domains
  – Dynamic call bindings
SPIN Architecture

• Protection model
  – A protection model controls the set of operations that can be applied to resources;
  – Pointers as capabilities\textit{(unforgeable references to resources)}
  – Questions (I have answers :-)
    • What are resources in SPIN?
    • Why is “pointers as capabilities” feasible?
    • How to deal with “externalized references”?
    • Does SPIN strictly disallow type casting?
SPIN Architecture

• Protection model
  – A protection domain defines the set of accessible names available to an execution context;
  – In SPIN, the Naming and protection interface is at the language level; (Ref: Figure 1)
  – How to manage namespace at the language level?
SPIN Architecture

• Protection model
  – Domain interface (or INTERFACE Domain)
    • Create
    • CreateFromModula
    • Resolve
    • Combine
(Ref: Figure 2)
SPIN Architecture

• Extension model
  – In SPIN, extensions are defined in terms of events and handlers;
  – A dispatcher is used to help install handlers, to pass arguments from events to handlers, and routes events to handlers;
  – An event’s authority (default implementation Modula) specifies an authorization procedure for operations that concern the event.
SPIN Architecture

• Questions (I have answers :-)

  Question: What are the syntactical forms for events and handlers? Why?

  Question: How to register an event handler? And how to incorporate an extension to SPIN? What’s the relation between to register handlers and to incorporate extensions? Are handlers added statically or dynamically?

  Question: When are type checking and dynamic linking performed?
SPIN Architecture

• Questions (I have answers :-)
  Question: What are “closures” and “guards” in SPIN?
    Why to have them? Can guards have side-effects?
    Why or why not? Are guards part of handlers?
  Question: Can multiple handlers installed on the same event be ordered? Who decides the order constraints?
  Question: How to handle results? Is the so-called “result handler” defined per-event or per-system?
  Question: How to deal with malicious handlers?
Core Services

- Memory management (Ref: Figure 3)
  - Physical address service
  - Virtual address service
  - Translation service
Core Services

• Thread management (Ref: Figure 4)
  – Strand
  – Exported procedures
    Block
    Unblock
    Checkpoint
    Resume
System Performance

• System size (Ref: Table 1)
• Micro-benchmarks (Ref: Table 2,3,4)
• Networking (Ref: Table 5,6 and Figure 5)
• End-to-end performance(Ref: Figure 6)
• Other issues(Ref: Table 7)
Summary

SPIN demonstrates that it is possible to achieve good performance in an extensible system without compromising safety using a programming language with appropriate features and software architecture principles.