Extensibility, Safety and Performance in the *SPIN* Operating System

Presented by Allen Kerr

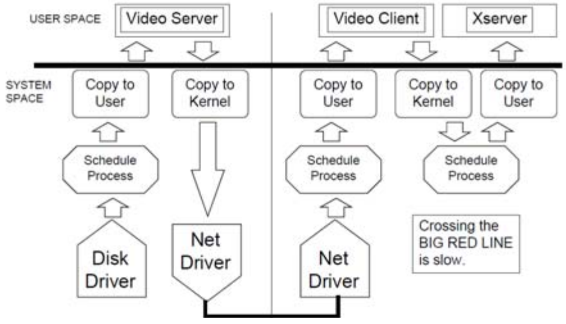
Overview

- Background and Motivation
- Modula-3
- SPIN architecture
- Benchmarks
- Conclusion

Hardware Vs Software Protection

- Hardware
  - □ One-size-fits-all approach to system calls
  - □ Requires software abstraction
- Software
  - □ Applications tell the system what needs to be done
  - □ Allows checks to be optimized using assumptions
  - □ Allows untrusted user code to be safely integrated into the kernel

How Network Video works



USER SPACE

Video Server

Video Client

Xserver

SYSTEM SPACE

Copy to User

Copy to Kernel

Copy to User

Copy to Kernel

Copy to User

Schedule Process

Schedule Process

Schedule Process

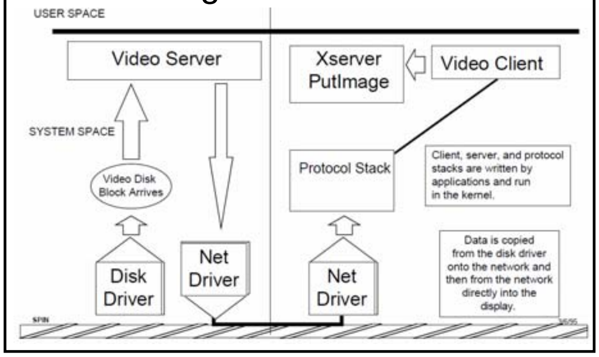
Disk Driver

Net Driver

Net Driver

Crossing the BIG RED LINE is slow.

How It Ought to Be



USER SPACE

Video Server

Xserver PutImage

Video Client

SYSTEM SPACE

Video Disk Block Arrives

Protocol Stack

Client, server, and protocol stacks are written by applications and run in the kernel.

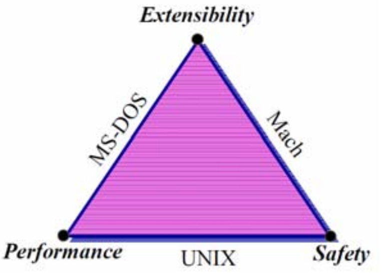
Disk Driver

Net Driver

Net Driver

Data is copied from the disk driver onto the network and then from the network directly into the display.

Motivation



Taken from talk “Language Support for Extensible Operating Systems”

Modula-3

- Similar feature set to Java
  - Pointer safety
  - Exceptions
  - Interfaces
  - Modules
  - Static Type Checking
  - Dynamic Linking
- Concerns
  - Execution Speed
  - Threads, allocation, and garbage collection
  - Memory Usage
  - Mixed-Language Environment

SPIN

- Kernel programmed almost exclusively in Modula-3
- Applications can link into kernel
- Examples of services
  - Filing and buffer cache management
  - Protocol processing
  - Scheduling and thread management
  - Virtual memory

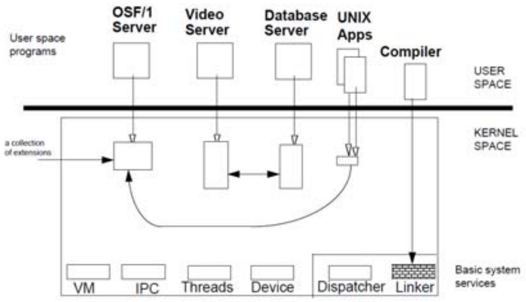
Further SPIN Motivation

- Most OSs balance generality with specialization
- General systems run many programs but run few well
- Specializing general operating system
  - Costly
  - Time consuming
  - Error-prone

Goals

- Extensibility
  - Allow applications to extend any service
- Performance
  - Dynamically inject application code into the kernel
- Safety
  - Rely on language protection for memory safety
  - Rely on interface design for component safety

SPIN System Components



Related Work

- Hydra
  - Applications manage resources
  - High overhead
- Microkernels
  - High communication overhead
- Software Fault Isolation
  - May lack necessary flexibility
- Aegis
  - Same goals as SPIN, different implementation

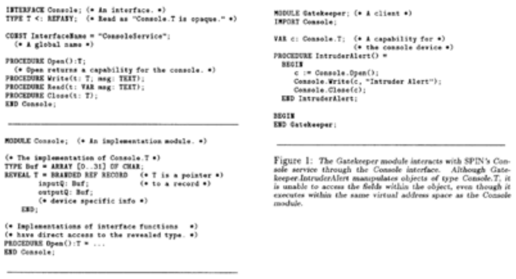
SPIN Architecture

- Co-location
  - □ Low cost communication between system and extensions
- Enforced modularity
  - □ Extensions written in modula-3
- Logical protection domains
  - □ Namespaces
- Dynamic call binding
  - □ Calls respond to system events.

Protection Model

- Defines a set of accessible names
- Language level protection
  - □ If you have the reference, you have access
- Code is safe if signed by a modula-3 compiler
- Create
  - □ Creates a new domain
  - □ Safe object file
  - □ Leaves imported interface symbols unresolved
- Resolve
  - □ Dynamic linking
  - □ Resolves undefined symbols
- Combine
  - □ Combines 2 existing domains

Example



```
INTERFACE Console; (* An interface. *)
TYPE T <: REFANY;  (* Read as "Console.T is opaque." *)

CONST InterfaceName = "ConsoleService";
  (* A global name *)

PROCEDURE Open():T;
  (* Open returns a capability for the console. *)
PROCEDURE Write(t: T; msg: TEXT);
PROCEDURE Read(t: VAR msg: TEXT);
PROCEDURE Close(t: T);
END Console;
```

```
MODULE Console;  (* An implementation module. *)

(* The implementation of Console.T *)
TYPE Buf = ARRAY [0..31] OF CHAR;
REVEAL T = BRANDED REF RECORD    (* T is a pointer *)
        inputQ: Buf;             (* to a record *)
        outputQ: Buf;
        (* device specific info *)
    END;

(* Implementations of interface functions   *)
(* have direct access to the revealed type. *)
PROCEDURE Open():T = ...
END Console;
```

```
MODULE Gatekeeper; (* A client *)
IMPORT Console;

VAR c: Console.T;  (* A capability for *)
                   (* the console device *)
PROCEDURE IntruderAlert() =
  BEGIN
     c := Console.Open();
     Console.Write(c, "Intruder Alert");
     Console.Close(c);
  END IntruderAlert;

BEGIN
END Gatekeeper;
```

Figure 1: *The Gatekeeper module interacts with SPIN's Console service through the Console interface. Although Gatekeeper.IntruderAlert manipulates objects of type Console.T, it is unable to access the fields within the object, even though it executes within the same virtual address space as the Console module.*

Extension Model

- Determines the ease, transparency and efficiency of extensibility
- Communication styles
  - □ Passive monitoring
  - □ Offer hints to the system
  - □ Replace current functionality
- Events
  - □ Announcement to the system
  - □ Request for service
- Handlers
  - □ Procedure that receives a message
  - □ Registered through central *dispatcher*
- Right to call procedure is equivalent to right to raise an event

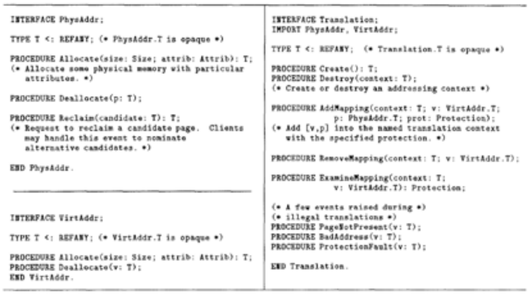
Core Service

- Kernel services that control hardware resources
  - □ Extensible Memory Management
  - □ Extensible Thread Management

Extensible Memory Management

- Three main interfaces
  - □ Physical Storage (Physical Addressing)
    - Use of pages
    - Allocation of pages
    - Controlled by core services
  - □ Naming (Virtual Addressing)
    - Bind to a process
    - Controlled by references
  - □ Translation
    - Raises exceptions related
- Does not implement memory management directly
- Provide base for higher levels

Memory management interfaces



```
INTERFACE PhysAddr;

TYPE T <: REFANY; (* PhysAddr.T is opaque *)

PROCEDURE Allocate(size: Size; attrib: Attrib): T;
(* Allocate some physical memory with particular
   attributes. *)

PROCEDURE Deallocate(p: T);

PROCEDURE Reclaim(candidate: T): T;
(* Request to reclaim a candidate page.  Clients
   may handle this event to nominate
   alternative candidates. *)

END PhysAddr.
```

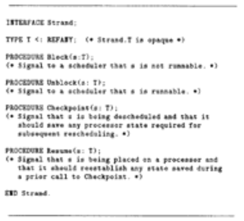
```
INTERFACE VirtAddr;

TYPE T <: REFANY; (* VirtAddr.T is opaque *)

PROCEDURE Allocate(size: Size; attrib: Attrib): T;
PROCEDURE Deallocate(v: T);
END VirtAddr.
```

```
INTERFACE Translation;
IMPORT PhysAddr, VirtAddr;

TYPE T <: REFANY;  (* Translation.T is opaque *)

PROCEDURE Create(): T;
PROCEDURE Destroy(context: T);
(* Create or destroy an addressing context *)

PROCEDURE AddMapping(context: T; v: VirtAddr.T;
                     p: PhysAddr.T; prot: Protection);
(* Add [v,p] into the named translation context
   with the specified protection. *)

PROCEDURE RemoveMapping(context: T; v: VirtAddr.T);

PROCEDURE ExamineMapping(context: T;
                     v: VirtAddr.T): Protection;

(* A few events raised during *)
(* illegal translations *)
PROCEDURE PageNotPresent(v: T);
PROCEDURE BadAddress(v: T);
PROCEDURE ProtectionFault(v: T);

END Translation.
```

Extensible Thread Management

- Applications can link their thread package
- No defined thread model
- Defines structure to build thread model on
  - Strands
  - Set of events
    - Block
    - Unblock
- Management only effects outside of kernel

Thread Interfaces



```
INTERFACE Strand;

TYPE T <: REFANY;  (* Strand.T is opaque *)

PROCEDURE Block(s:T);
(* Signal to a scheduler that s is not runnable. *)

PROCEDURE Unblock(s: T);
(* Signal to a scheduler that s is runnable. *)

PROCEDURE Checkpoint(s: T);
(* Signal that s is being descheduled and that it
   should save any processor state required for
   subsequent rescheduling. *)

PROCEDURE Resume(s: T);
(* Signal that s is being placed on a processor and
   that it should reestablish any state saved during
   a prior call to Checkpoint. *)

END Strand.
```

Figure 4: *The Strand Interface. This interface describes the scheduling events affecting control flow that can be raised within the kernel. Application-specific schedulers and thread packages install handlers on these events, which are raised on behalf of particular strands. A trusted thread package and scheduler provide default implementations of these operations, and ensure that extensions do not install handlers on strands for which they do not possess a capability.*

Implications for Trusted Services

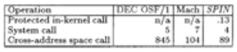
- Core services interact with hardware
- They must follow their specifications
- Trust is required for extension building

System Performance

- System Size
  - Measured by lines of code and object size
- Microbenchmarks
  - Low level system services
- Networking
  - Suite of network protocols
- End-To-End Performance
  - Show performance of two applications

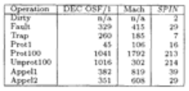
Microbenchmark results

- Shows a significant performance increase



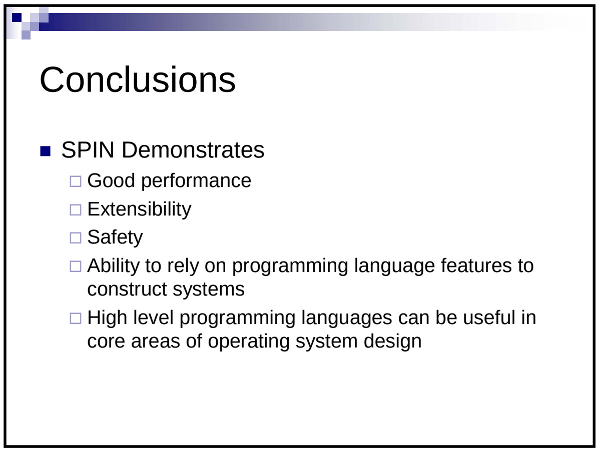
Operation	DEC OSF/1	Mach	*SPIN*
Protected in-kernel call	n/a	n/a	.13
System call	5	7	4
Cross-address space call	845	104	89

Table 2: *Protected communication overhead in microseconds. Neither DEC OSF/1 nor Mach support protected in-kernel communication.*



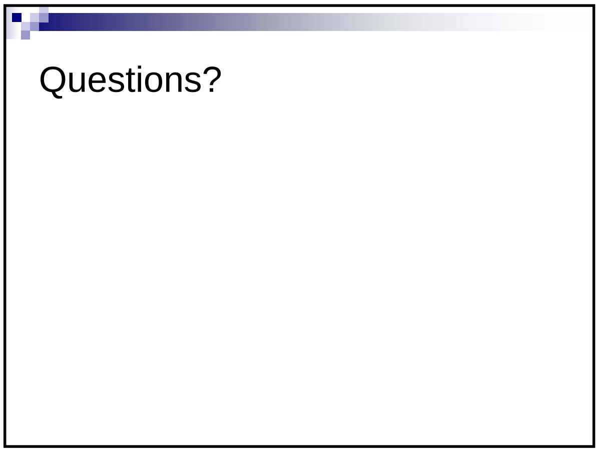
Operation	DEC OSF/1	Mach	*SPIN*
Dirty	n/a	n/a	2
Fault	329	415	29
Trap	260	185	7
Prot1	45	106	16
Prot100	1041	1792	213
Unprot100	1016	302	214
Appel1	382	819	39
Appel2	351	608	29

Table 4: *Virtual memory operation overheads in microseconds. Neither DEC OSF/1 nor Mach provide an interface for querying the internal state of a page frame.*

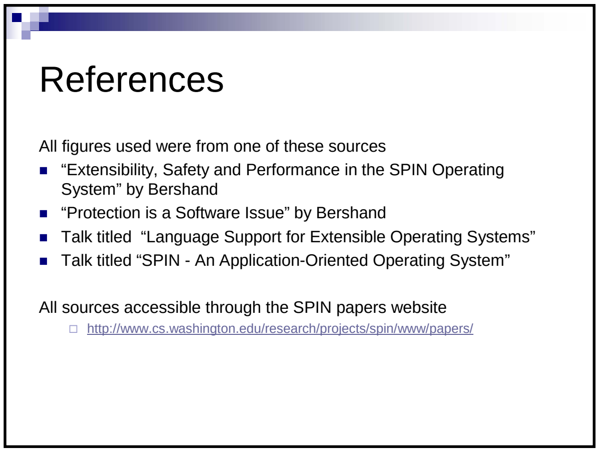


Conclusions

- SPIN Demonstrates
  - □ Good performance
  - □ Extensibility
  - □ Safety
  - □ Ability to rely on programming language features to construct systems
  - □ High level programming languages can be useful in core areas of operating system design



Questions?



References

All figures used were from one of these sources

- “Extensibility, Safety and Performance in the SPIN Operating System” by Bershand
- “Protection is a Software Issue” by Bershand
- Talk titled “Language Support for Extensible Operating Systems”
- Talk titled “SPIN - An Application-Oriented Operating System”

All sources accessible through the SPIN papers website

- □ http://www.cs.washington.edu/research/projects/spin/www/papers/