

Exokernel & SPIN OS

Same Motivation

General operating system is not suitable for all application

Memory management/Buffer Management/Thread Model etc.

Different Design

SPIN: In-kernel extension – A single kernel virtual address space

Benefits: Extensibility; Performance.

Problems: Safety?

Crash/Unstable/Lack of Isolation?

OS Structure

Application Layers

More Services

Core Services

Abstract of Hardware Resource

Hardware

Question:

Where to draw the kernel/user boundary line?

SPIN's choice

Several factors affect SPIN's choice:

- *User/Kernel Communication cost.*
- *Export Interface*

SPIN's Structure

- Single kernel address space;
- Multiple components communicate through well defined interface
- Extensibility through new module

Integrity of the Kernel

Common methods for safely extension

- **Utilize Hardware Support**
Intel 4 ring levels, Memory segment ...
- **Little language**
Being little; Interpretation cost;
- **Software Fault Isolation**
Binary rewrite: jump/call/memory access;
Lack of a protection model: it's only a mechanism .
- **Language level guarantee**
SPIN uses Modula-3 language to implement the extension model.

Modula-3 Properties

Define of Module

- **The Visible Parts**

Interface: How you access the *module*.

- **The Hidden Parts**

Implementation; Status; etc.

Type Safe

- **Controlled Access**

No type-casting allowed;

Check for Bounds Violation of Array Indexing OP.

Automatic Storage Management

Protection Model Basic

- **Capability based:**
Everything has to be accessed through a reference including
 - System objects (physical pages, device buffers etc.)
 - Interfaces (different implementations)
- **Using Pointers for References**
 - Unforgeable
 - Type safety; Automatic storage management
 - Externalized reference using indexed table
 - Application not written in Module-3

Examples

```
INTERFACE Console;  
  TYPE T <: REFANY;  
  CONST InterfaceName = "ConsoleService"  
  PROCEDURE Open():T;  
  PROCEDURE Write(t:T; msg:TEXT);  
  ...  
END Console;  
  
MODULE Console;  
  TYPE Buf=...  
  PROCEDURE Open():T =...  
END Console;
```

```
MODULE Gatekeeper;  
  IMPORT Console;  
  
  VAR c:Console.T;  
  PORCEDURE IntruderAlert()=  
  BEGIN  
    c:=Console.Open();  
    Console.Write(c,...);  
    Console.Close(c);  
  END IntruderAlert;  
  ...
```

Protection Domains

A logical space to organize named-interfaces

- A set of names/program symbols can be referenced by code with access to the domain
- A set of safe object files with exported interfaces.
- RESOLVE procedure: dynamic linking
Procedure call level overhead once resolved.
Intersecting domains for sharing services.
- COMBINE procedure: union of exist domains.

Extension Model

Events

A message that announces system state change, or a request for service
Indistinguishable from a procedure defined in an interface.

Handlers

Procedures of the same type that receives the message.

Any number of handlers can be associated with an event

Single handler → procedure call

Multiple handlers → dynamically compiled dispatch routine

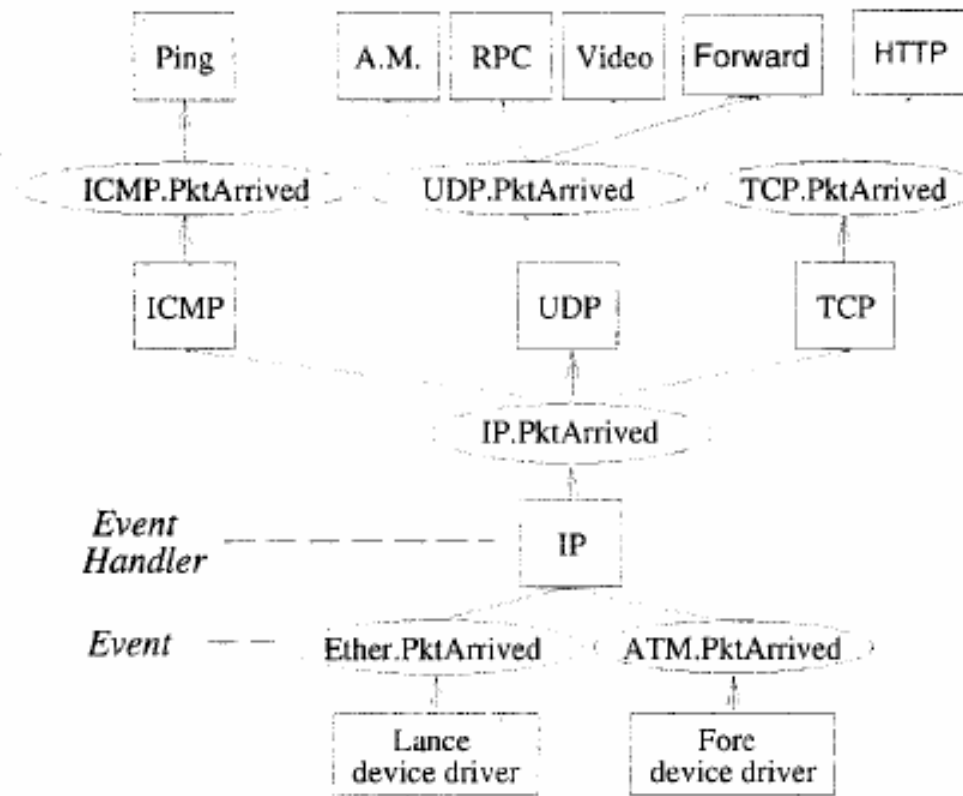
Extension Module

A primary module statically exports the procedures – primary handlers

Other modules interact with primary module

Can deny/accept the handler; associate guards; restrict execution

NetWorking



Core Service: Memory

Extensible Memory Management:

3 basic components

- Physical Storage (Physical Address): Allocate/Deallocate/Reclaim
- Naming (Virtual Address): Allocate/Deallocate
- Translation (Mapping): AddMapping/RemoveMapping/...
PageNotPresent/BadAddress/ProectionFault

Examples:

*Return an alternative page when Reclaim
CopyOnWrite on ProectionFault etc. events.*

Core Service: Thread

Strands

Structures on which a thread model rests.

Competing contexts among which the scheduler multiplex processing resource

Event

Block/Unblock/Checkpointing/Resume

Application Specific Scheduler

An special implementation of strand

Utilize Checkpointing/Resume to schedule its own strands.

Performance Evaluation

Size/Complexity

Microbenchmarks

- Protected Communication
- Thread Management
- Virtual Memory Management

Networking

Applications

Conclusion

- Co-location: *Single kernel address space*
- Enforced modularity: *special language*
- Logical protection domain
- Dynamic call binding
event/handler mechanism using procedure calls

Questions?

If you want the class overload permit, and have not got it from Karsten yet, please write down your name/gtid and give it to me after class.