Operating Systems Organization, Con't

CS 239, Spring 1995

June 1, 1995

Organizing the Total System

In microkernel organizations, much of the OS is outside the microkernel

But that doesn't answer the question of how the system as a whole gets organized

How do you fit together the components to build an integrated system?

While maintaining all the advantages of the microkernel

A Sample Microkernal OS Organization - Spring

Developed by Sun

Intended to examine how to improve OS's by building from the ground up

Approach was to address the greatest problems in building operating systems

In some senses a response to problems with Unix

Unix Problems Spring Addresses

- Cost of maintaining/evolving system
- Inflexible security model
- Hard to build distributed services
- Hard to handle real-time issues
- Multiplicity of naming schemes

Basic Spring Approach

Make it possible for others to extend the OS itself

Which requires strong interfaces

Which are open, flexible, and extensible

Spring designers clearly learned from success of extensible Unix features (like VFS)

This desire lead to use of object oriented approach

Object Oriented Organizations

Object oriented organization is an increasingly popular way to build software

Well suited to OS development, in some ways

- OS's manage important data structures
- OS's are modularizable
- Strong interfaces are good in OS's

Object Orientation and Extensibility

- One of the main advantages of object oriented programming is extensibility
- Operating systems increasingly need extensibility
- So, again, object oriented techniques are a good match for operating system design

How "Object Oriented" Should an OS Be?

- Many OS's have been built with object oriented techniques
  - E.g., Mach and Windows NT
- But most of them leave object orientation at the microkernel boundary
  - No attempt to force object orientation on out-of-kernel modules

Spring is a Microkernel System

- Spring microkernel consists of nucleus and basic virtual memory support
- Nucleus supports operating system objects
  - Secure
  - High speed object invocation

Spring Object Managers

- Other than microkernel, Spring is implemented as a suite of object managers
  - Running in non-kernel mode
  - But in different address spaces than user programs
- Adding new functionality to Spring amounts to adding a new object manager
- Object managers are objects, themselves

Spring's Interface Definition Language

- Spring wants to avoid being tied to a single language
- But it also requires strong interfaces to allow for extensibility
- So Spring interfaces are written in IDL
- Interfaces are defined in IDL, but IDL says nothing about implementations

IDL Compilers

- Convert IDL definitions of interfaces into particular languages
- To generate language-specific form of the interface for use of objects written in that language
- Also generates client and server stub code for use by objects employing the interface

Objects in Spring

Object users invoke operations defined in its interface

The operation could be performed in

- the same address space as the called
- a different address space on the same machine
- a different address space on a different machine

Server-Based Objects

Server-based Spring objects live in separate address spaces from their clients

- Stub generation from IDL for their benefit

Subcontracts and *doors* used to communicate between clients and servers

- Essentially, passed to another address space by a pointer

Serverless Objects

Objects kept in the caller's address space

Typically used for lightweight objects mostly of local interest

Can be passed to another address space by copying

Parts of a Spring Object

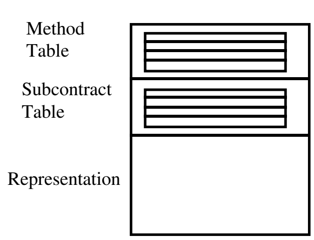
From the client's point of view, an object consists of -

1). A method table

2). A subcontract operation vector

3). Client-local private state (*representation*)

Spring Object Diagram



Methods and Spring Objects

Spring object methods are either handled in the object's local address space

- Through the method table

Or in a remote address space

- Through the subcontract table

Subcontracts essentially allow other objects to handle your methods for you

Spring Subcontracts

Semantics of invoking an object in a different address space can vary
- Can the object be replicated?
- Can it support an atomic transaction?
- Can it migrate?
- Is it persistent?

Spring subcontracts allow this flexibility

In the context of RPC

Subcontracts and Extensibility

Subcontracts are essentially an extensibility mechanism

They allow service providers to extend the service

Without requiring clients to do things differently

Essentially, subcontracts sit between interfaces and implementations

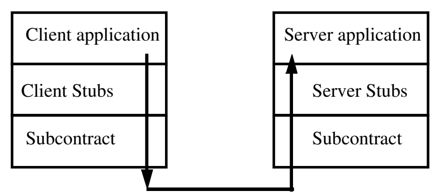
Simple Subcontracts

One example is a subcontract for invoking a method on an object at a remote server

Subcontract implements the machinery for communicating with the remote server

Methods simply marshal arguments and call the subcontract, in this case

Simple Subcontract Diagram



So What Can I Do With Subcontracts?

One example - a simple replication service

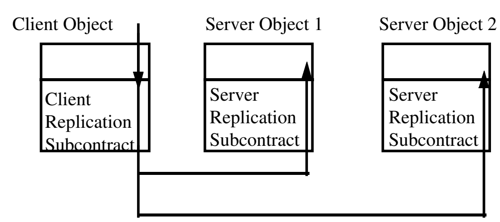
Users access through client object

Server objects maintain replication

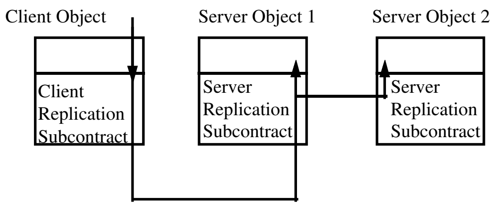
Client object has representation showing where each server maintaining a replica is

All local methods are stub calls to subcontracts

Replication Subcontract Diagram



Or . . .



Other Types of Subcontracts

- The simplex subcontract - supports simple invocation of a single remote object
- The cluster subcontract - supports multiple objects supported by single server
- The caching subcontract - supports access to either remote object or local caching object

Spring Nucleus Abstractions

- Domains
- Threads
- Doors

All used to support Spring's basic object model

Spring Domains

Provide address space and container to hold application resources

Similar to Unix processes

Or Mach tasks

Spring Threads

Unit of execution in Spring

Similar to threads in other systems

Spring domains are typically multithreaded

Spring Doors

Abstraction supporting interdomain object-oriented method calls

A door describes an entry point to a domain

Also like a capability

Possession of a door implies right to invoke an object's method

Protecting Doors

Since doors are capabilities, kernel must protect them to provide security

Domains don't hold doors themselves -

They hold *door identifiers*

Door identifiers point to doors stored in the kernel

Kernel maintains per-domain door table

Obtaining Doors

Only two ways for a domain to get a door -

1). From the domain that the door opens to

2). From another domain that already has it

Target domain can't tell who used a door

Cross-Domain Object Invocation Via Doors

Client invokes door via door identifier

Nucleus allocates a server thread in target domain and quickly transfers control to it

Passing door information and arguments

When target wishes to return, nucleus deactivates the called thread, reactivates the caller thread, passes return data

Door Invocation Methods

Kernel supports three flavors of door invocation

1). The fast-path

2). The vanilla-path

3). The bulk-path

Stubs make choice invisible to user, typically

The Fast-Path Door Invocation

For simple data values, less than 16 bytes

Which is the dominant case

No doors may be passed

Highly optimized - around 100 Sparc instructions to cross domains and come back

The Vanilla-Path Door Invocation

For passing less than 5 Kbytes of data

Including moderate number of doors

Data passed through the kernel

The Bulk-Path Door Invocation

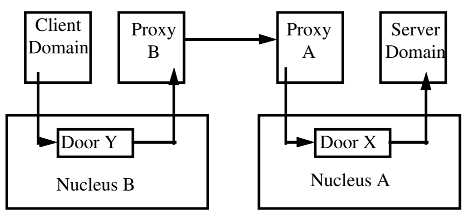
For sending entire pages of data

- And/or large numbers of doors

Uses VM remapping to move data

- Can either unmap and remap in target address space
- Or map into both and use copy-on-write

Network Proxy Diagram



Virtual Memory In Spring

Each Spring machine has one Virtual Memory Manager (VMM)

VMM handles mapping, sharing, page protection, transfers, and caching of local memory

External pagers access backing store

Spring Network Proxies

When a door points to an off-machine object, it actually points to a network proxy

Network proxies effectively connect multiple Spring machines

Network proxies are user-mode server domains

Proxies are per-protocol, not per-machine

Spring Security

Doors provide some level of security

- But clearly are lacking in certain ways

Augmented with both access control lists and capabilities

Essentially works on the basis of putting a security object in front of the real object

- Security object can check capability, ACL

Address Space Objects

Represents the virtual address space of a Spring domain

Implemented by VMM

Represents just the address space, not particular pieces of real memory (either in terms of physical memory or memory actually containing data)

Memory Objects

Abstraction of memory that can be mapped into an address space object

Memory objects represent memory

Implemented by some object at the user level

Operations include set/query length and bind

Not page-in/page-out - separate object provides paging

Cache and Paging Objects

Pager objects know how to fetch and store pages of an object

Pager objects provides methods to actually fetch pieces of memory

VMM provides cache objects that actually control pages of physical memory

Cache/Pager Communications

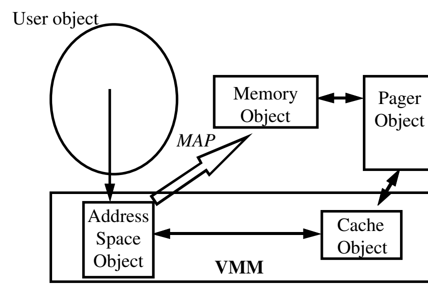
Caches are where the pages are stored in a way available to the domain that wants them

Pagers know how to get the pages from where they're normally kept

Cache ask pagers for pages

Pagers tell caches what to invalidate

Virtual Memory Object Diagram



What's the Point of All This?

- Separate address space object allows a domain's address space to be implemented by multiple different objects
- Separate memory objects allows flexibility in handling memory (e.g., two different memory objects can access the same memory)
  - Also allows "stackable" file system ops

What's the Point of All This, Con't

- Separate pager objects required to actually get pages of data
- Separate cache objects required to actually handle the physical memory and virtual memory hardware of the machine

The Joys of Flexibility

Pagers can fetch pages from disk
- Or across the net

Different memory objects can permit different types of access to the same memory
- E.g., read-only versus read/write

A single address space object can have memory provided from mapped files, normal VM, off-site files

More Joys of Flexibility

Since the address space object is a normal Spring object, we can create doors to it
- So other objects (even in other domains) can access it

Since multiple pagers are possible, they can be optimized for their backing store
- Such as a log-based file system versus an extent-based file system

Another Example - Distributed Shared Memory

No problem - let multiple address space objects on different machines map in the same memory objects

Pagers then provide access to data
- And enforce coherency

What This Really Means

Virtual memory, shared memory, distributed shared memory, file systems, stackable file systems, caches, everything -

Provided by one set of interoperable mechanisms

Extreme power, extreme flexibility!

Naming in Spring

Spring uses a single unified system to name all resources

Any object can be bound to any name

And objects of different types can share the same name space

Contexts in Spring

Names are bound to objects within a *context*

Contexts are objects containing set of *name bindings*

All naming operations go through contexts

Contexts can be bound inside other contexts
- Allowing connection of name spaces in a *naming graph*

Naming and Persistence

By default, Spring objects are not persistent

Binding an object to a name in a persistent namespace makes the object persistent

- Also provides method of re-obtaining it

How does a name context make an arbitrary object persistent?

Each object type provides implementation of persistence (through a general interface)