L06a. Spring Operating System

Introduction:

- How to design for the continuous and incremental evolution for the complex distributed SW systems (Functionality + Performance)?
  - Using Distributed Object Technology, we can design scalable OS services in a parallel system.
- How to innovate in the commercial OS space?
  - In a commercial space, the choice becomes whether to build a new OS or to build a better implementation of an existing OS. Due to the installed base of applications, moving to a new OS is not a good option. Sun, therefore, took the latter approach with Spring.
- The OS Sun was developing had to:
  - Run existing applications unchanged and yet allow for them to be scaled to larger capacities via clusters.
  - Allow for 3rd party vendors to use the APIs exposed optimally without breaking anything.

Object-Based vs. Procedural Design:

- Procedural Design:
  - The code is written as a monolithic entity.
  - We have a shared state represented by the global variables, and private states represented by the function calls.
- Object-Based Design:
  - The state is entirely contained inside the object.
  - Methods inside the object manipulate the state. Only the methods are visible externally.
- In Spring, Object Orientation is applied to build the OS Kernel.

Spring Approach:

- Provide strong interfaces to each sub-system.
- The system is open and flexible. Thus, 3rd party vendors can integrate their SW to the system.
- IDL (Interface Definition Language) is used to define the OS interfaces. This provided flexibility in terms of the programming language.
- A microkernel was used to facilitate extensibility:
  - Nucleus: Threads + IPC.
  - VM Manager: Memory management.
- Sun provided a “Network Proxy” node so that the OS can be used as a network OS.

Nucleus:

- Nucleus is the microkernel of Spring OS. It manages only the threads abstractions and the IPC.
- Door:
  - A SW capability to a target domain (a domain is container or an address space where threads can execute).
  - Threads can enter a target domain using the Door. It represents the entry point methods to the domain.
  - Every domain will have a Door Table containing the IDs of the Doors this domain can access.
  - Multiple client domains can have access to the same Door.
- How to make a Protected Procedure Call?
  - The client domain invokes the Door handle of the target domain.
  - The Nucleus verifies this invocation.
  - The Nucleus allocates a server thread to the target domain and executes the invocation that is indicated by this particular Door handle.
  - Since it's a Protected Procedure Call, the client thread will be deactivated, and the thread is allocated to the target domain to execute the invocation.
  - On return, the thread is deactivated, and the client thread is reactivated to continue execution.
  - Similar to LRPC in terms of doing fast cross address space calls using the Door mechanism.
  - This ensures high performance.

Object Invocation Across the Network:

- Object invocation between client/server domains across the network is facilitated using the Network Proxy.
- Each domain has multiple Network Proxies to communicate with different nodes on the network. This facilitates specialization since these different proxies can represent different protocols.
- Network Proxies are invisible to the clients/servers.
- How to establish communication over the network?
  - A Network Proxy ($Proxy_s$) will be instantiated on the server node and a Door will be created for communication between this Proxy and the server domain through the Nucleus.
  - $Proxy_s$ will export the network handle embedding the Door created on the server domain to the Network Proxy of the client domain $Proxy_c$. These Proxies and the communication between them are outside the kernel.
  - $Proxy_c$ will use this network handle to establish a connection between the client's Nucleus and the server's Nucleus.
  - The client domain will then invoke the Door inside its Nucleus and the two Proxies will facilitate the invoking the server.

Secure Object Invocation:

- Spring OS facilitates providing different privilege levels for different clients using the “Front Object”.
- An underlying object may have a Front Object that is directly connected to it (without Doors).
- Whenever a client domain tries to access this protected domain, the Front Object will check the Access Control List (ACL) to see what privileges this client domain has for accessing the protected domain.
- Different instances of the Front Object can be created with different access policies to the same underlying object.
- Doors are SW capabilities, which means they can be passed from client domain to another. When passing a Door, the client domain can determine whether to pass the same privileges or not.

Virtual Memory Management:

- Virtual memory management is part of the Spring OS kernel.
- The VMM is responsible for managing the linear address space of each process.
- The VMM breaks this linear address space into memory regions (sets of pages with different sizes).
- These memory regions are mapped to different memory objects.
- These memory objects are the mechanism by which parts of the address space can be mapped into different backing entities (e.g. desk, file system, etc.).
- Pager Object: Responsible for establishing the connection between virtual memory and physical memory. The Pager Object makes sure that a memory object has a representation in the physical memory.
- The Pager Object will create a cache object representation for the memory object in the DRAM.
- The Pager Object gives the ability to have different
- These associations between memory objects and physical objects can be dynamically created.
- The same memory object can have distinct cache representations on different physical memory spaces.
- The Pager Objects maintains the coherence of these different cache objects if needed.

Dynamic Client-Server Relationships:

- The client-server interaction should be irrelative to the physical locations of the clients and the servers (Same machine – different nodes on a network).
- Scenario #1: We have several replica of the servers (to increase availability). The clients will be dynamically routed to different servers depending on the physical proximity of the client and the server, and also on the load distribution.
- Scenario #2: The servers are cached. The clients can dynamically access these cached copies instead of accessing the servers themselves.
- Then, we have two types of dynamic decisions:
  - In Scenario #1: Deciding which server replica the client should access.
  - In Scenario #2: Deciding which cached copy the client should access.

- Spring OS uses “Subcontracts” to manage these decisions.

Subcontract:

- As mentioned earlier, IDL is used to define the OS interfaces.
- The Subcontract is the implementation of the IDL interfaces.
- This makes the Subcontract hide the runtime behavior of an object from the actual interface.
- The client stub generation will be simple, since all the details about the server (Where is the server? How to access it? Which replica/cached copy of the server should I go to?) will be hidden in the Subcontract.
- A Subcontract can be changed dynamically while the stub stays fixed. This facilitates adding functionalities to existing services using the Subcontract mechanism.
- Marshal/Un-marshal Interface: The Subcontract will marshal/un-marshal arguments when requested by the client. The Subcontract will do the appropriate steps to execute the request based on the location of the server (e.g. on the same machine, on the network, on a different processor, etc.).
- The Subcontract on the client side has an invocation mechanism.
- On the server side, the Subcontract has (create – revoke – process) mechanisms.

Conclusion:

- Spring OS uses object technology as a structuring mechanism in building a network OS.
  - Strong interfaces.
  - Flexibility.
  - Extensibility: Spring OS has a microkernel.
- Spring OS uses Network Proxies: The client and the servers don’t have to know the locations of each others.
- Subcontract mechanism allows the clients and the servers to dynamically change the relationships between each others without changing the client/server stubs.