**CMPSCI 677 Operating Systems** **Spring 2022**

# Lecture 3: February 2

*Professor: Prashant Shenoy* *Scribe: Rajul Jain (2022)*

## 3.1 Message-Oriented Communication

### 3.1.1 Communication between processes

Suppose two processes running on same machine want to communicate with eachother. There are 2 ways to achieve this.

**Unstructured Communication:**

- Processes use a shared memory/buffer or a shared data structure for communication.
- This type of communication is called "unstructured" because the buffer is just a piece of memory and has no structure associated with it. In other words, we can put any data that we want in the buffer and the processes have to interpret that data and do something with it.

**Structured Communication:**

- Processes send explicit messages to communicate, i.e., using Inter-Process Communication (IPC).
- This can be achieved by either low-level socket-based message passing or higher-level remote procedure calls.

Now suppose the two processes are running on different machines in a distributed system. If the processes communicate via unstructured communication, then we need to figure out a way to share buffers across machines. Clearly, making memory accessible across machines connected over a network is more difficult than using shared memory in the same machine. Hence, we prefer structured communication like sockets and RPCs.

**Question:** Why do distributed systems need low-level communication support for both structured and unstructured communication?

**Answer:** Because distributed systems are spread over a network.

### 3.1.2 Communication (Network) Protocols

Suppose two processes (App 1 and App 2), each using the OSI or TCP/IP model, communicate over a network. App 1 creates a message at the application layer which travels down the network stack layers (where each layer adds their headers to the original message) all the way to the physical layer. The message is then passed over to App 2's physical layer where it travels all the way up the network stack again to complete the communication.

### 3.1.3 Middleware Protocols

In a distributed system, *middleware* is the layer residing between the OS and an application. The middleware layer sits between the application and transport layer in the network protocol stack.

### 3.1.4 TCP-based Socket Communication

TCP-based socket communication is a structured communication method which uses TCP/IP protocols to send messages. The socket creates network address (IP address and port number) for communication. The socket interface creates network end-points. For example, in a client-server distributed system, both the client and the server need to create sockets which are bound to port number where they can listen to and send messages.

#### 3.1.4.1 Understanding TCP Network Overheads

Consider a client-server model using TCP method for communication. As TCP method follows a 3-way handshake protocol for communication, this leads to a transfer of 9 messages for sending a single request and subsequently receiving a single response. This is a huge overhead.

### 3.1.5 Group Communication

In a distributed system, when one machine needs to communicate with many machines, group communication protocols are used. This is analogous to sending an email to multiple recipients. For group communication, all the recipients subscribe to a *group address*. The network then takes care of delivering the messages to all the machines in the group address. This is called *multicasting*. If the messages are sent to all the machines in the network, then the process is called *broadcasting*.

## 3.2 Remote Procedure Calls

RPCs provide higher level abstractions by making distributed computing look like centralized computing. They automatically determine how to pass messages at the socket level without requiring the programmer to directly implement the socket code. In other words, instead of sending message to the server to invoke a method X, the programmer can call the method X directly from the client machine. RPCs are built using sockets underneath. The programmers do not write the this socket code. Instead, it is auto-generated by the RPC compiler. Stubs are another piece of RPC compiler auto-generated code which convert parameters passed between client and server during RPC calls.

There are a few semantics to keep in mind:

- Calling a method using RPC is same as invoking a local procedure call.
- One difference is that in an RPC, the process has to wait for the network communication to return before it can continue its execution, i.e., RPCs are *synchronous*.
- *You cannot pass pointers or references.* Pointers and references point to a memory location in the respective machine. If these memory locations are passed on a different machine, they will point to something completely different on that machine.

- *You cannot pass global variables.* As global variables lying on one machine cannot be accessed by another machine simply over a network. Hence global variables are not allowed in an RPC.
- *Pass arguments by value.*

**Question:** If we absolutely need to pass pointers in an RPC, how do we achieve that?

**Answer:** Take the entire object (say, from the client machine) and pass it on to the server. Create a copy of this object on server and then create a local pointer to the object.

### 3.2.1 Marshalling and Unmarshalling

Different machines use different representation of data formats. This creates discrepancy in understanding messages and data between different machines. Hence before sending messages to the other machine, the messages are converted into a standard representation such as eXternal Data Representation (XDR). Marshalling is the process of converting data on one machine into a standard representation suitable for storage/transmission. Unmarshalling is the process of converting from a standard representation to an internal data structure understandable by the respective machine.

### 3.2.2 Binding

The client locates the server using bindings. The server that provides the RPC service registers with a naming/directory service and provides all of the details such as method names, version number, unique identifier, etc. When client needs to access a specific method/functionality, it will search in the naming/directory service if there's a service in the network which hosts this method or not and then accesses the server using the IP and port number listed in the directory.

## 3.3 RPC Implementation

### 3.3.1 Failure semantics

- Lost request/ response messages: The network protocols handle these errors by timeout mechanisms.
- Server and client failures: Need to be handled while creating distributed systems.

If client crashes after sending a request to the server, then the server response is referred to as **orphan**. To deal with the orphans, methods such as Extermination, Reincarnation, Gentle reincarnation, and expiration can be used.

### 3.3.2 Case study: SUNRPC

SUNRPC was developed for use with NFS. It is one of the widely used RPC systems. Initially, it was built on top of UDP, but later transferred to TCP. It uses SUN's XDR format.