Innopolis University

Distributed Systems Reading Questions 3

Timur Samigullin

supervised by
Azat Safin

November 4, 2016

Question 1: How are the OSI model layers mapped to the adapted (middleware-centric) reference model? Discuss in terms of functionality.

- The highest layer in both approach is application layer. Mostly it works with applications like HTTP, SMTP, FTP, TELNET and others.
- The session and presentation layer of OSI replaced with signle middleware layer in middleware-centric model, it contains application-independent protocols. Protocols at these layers provide a connections between low-level protocols and application layer.
- Network and transport layer corresponds to operating system layer, its provide communication services. In general networks, routing and packets transferring is performed at this level, transport level is operate with ports and TCP/UDP protocols.
- Hardware layers in OSI models, which are physical and data-link layers replaced with one hardware layer. These layers operate with electrical signals, that transfers through wires or other communication channels.

Question 2: Give some examples of general-purpose protocols, which belong to the middleware layer. What types of distribution transparency do they provide?

General-purpose protocols are protocols, that are useful to many applications, but which cannot be qualified as transport protocols. For example, DNS, it provides general-purpose, application independent service. Another example are various ways to establish authentication, that is provide proof of a claimed identity. Distributed commit protocols also can be correlated to this group of protocols, it can be present an interface independently of specific applications, thus providing a general-purpose transaction service. The last example is distributed locking protocol, by which a resource can be protected against simultaneous access by a collection of processes that are distributed across multiple machines.

All these protocols used for realizing access transparency, because in general way, users will not know about using of this protocols, for them it is transparent.

Question 3: Give some examples of systems evolving from combining persistent/transient, synchronous/asynchronous and discrete/streaming communication. Which combination corresponds to RPC?

- Persistent communication: Messages are stored until (next) receiver is ready, example is email.
- Transient communication: Message is stored only so long as sending/receiving application are executing. Transport-level communication services offer transient communication
- Asynchronous communication: Sender continues immediately after it has submitted the message.
- Synchronous communication: Sender blocks until message is stored in a local buffer at the receiving host or actually delivered to sending.
- In distinction communications the parties communicate by messages, each message forming a complete unit of information.
- Streaming communication involves sending multiple messages, one after the other, where the messages are related to each other by the order they are sent.

The example of transient asynchronous communication is UDP, because this protocol is not control delivery status of message, it means that program will continue execution after sending message. The example persistent asynchronous communication is email. RPC it is a way to allow programs to call procedures located on other machine. The RPC is coresponds with transient communication with synchronization after the request has been fully processed is also widely used.

Question 4: Can we simply use pointers as parameters in RPC calls without extra care? Why?

We can use pointers in RPC, but in some different way. A pointer is meaningful only within the address space of the process in which it is being used. they make only locally sense: they refer to memory locations that have meaning only to the calling process.

Problems can be alleviated by using global references: references that are meaningful to the calling and the called process. For example, if the client and the server have access to the same le system, passing a le handle instead of a pointer may do the trick. There is one important observation: both processes need to know exactly what to do when a global reference is passed. In other words, if we consider a global reference having an associated data type, the calling and called process should have exactly the same picture of the operations that can be performed. Moreover, both processes should have agreement on exactly what to do when a file handle is passed. So, this issue can be solved by proper programming-language support.

Question 5: In multicast RPC, should the client wait for all responses? Why?

It depends of purpose implementing multicast RPC. When the server has been replicated for fault tolerance, we may decide to wait for just the first reponse, or perhaps until a majority of the servers returns the same result. On the other hand, if the servers have been replicated to do the same work but on different parts of the input, their results may need to be merged before the client can continue.

Question 6: Consider implementing an application using RPC on top of streams (TCP) and datagrams (UDP) sockets, respectively. What would be your main challenge?

RPC actually operates over UDP or TCP. RPC/UDP is a connectionless, stateless protocol. RPC/TCP is slower, but provides a reliable, stateful connection. We can use UDP on local LAN, when we can fall back on rather simple mechanisms for timeout and retransmission, then with TCP handshake.

Question 7: What are the differences between ZeroMQ and Berkeley (traditional) sockets? What is the side effect of asynchronous connection-oriented communication? What are the main communication patterns supported by ZeroMQ?

ZeroMQ sockets can support many-toone communication instead of just one-toone communication as is the case with standard Berkeley sockets.

A side effect of combining asynchronous with connection-oriented communication, is that a process can request a connection setup, and subsequently send a message even if the recipient is not yet up-andrunning and ready to accept incoming connection requests, let alone incoming messages.

Pair of sockets, that using for sending and receiving data is called pattern. The three most important communication patterns supported by ZeroMQ are request-reply, publish-subscribe, and pipeline.

- The request-reply pattern is used in traditional client-server communication, like the ones normally used for remote procedure calls. The request-reply pattern simplifies matters for developers by avoiding the need to call the listen operation, as well as the accept operation. Moreover, when a server receives a message, a subsequent call to send is automatically targeted toward the original sender. Likewise, when a client calls the recv operation (for receiving a message) after having sent a message, ZeroMQ assumes the client is waiting for a response from the original recipient.

- In the case of a publish-subscribe pattern, clients subscribe to specic messages that are published by servers, only the messages to which the client has subscribed will be transmitted.If a server is publishing messages to which no one has subscribed, these messages will be lost.
- the pipeline pattern is characterized by the fact that a process wants to push out its results, assuming that there are other processes that want to pull in those results.

Question 8: What is the role of the message brokers? Why/when are they needed?

When two application communicate via messages, it should clearly understand each other. In practice, it means, that applications should communicate with the same protocol. The problem of distributed system is the difference between communication protocols of two applications due to different reasons. The reason is to provide some gateway to make communication between applications. In message-queuing systems, conversions are handled by special nodes in a queuing network, known as message brokers. A message broker acts as an application-level gateway in a message-queuing system. Its main purpose is to convert incoming messages so that they can be understood by the destination application. In a messagequeuing system, a message broker is just another application.

At basic level message-brokers can be just simple reformatter for a messages, in more advanced level a message broker may act as an applicationlevel gateway, in which information on the messaging protocol of several applications has been encoded.