Theoretical exercises of Chapter 4: Communications

Đào Ngọc Thành - 20163693

Question 1: In many layered protocols, each layer has its own header. Surely it would be more efficient to have a single header at the front of each message with all the control in it than all these separate headers. Why is this not done?

Each layer must be independent of the other ones. The data passed from layer k + 1 down to layer k contains both header and data, but layer k cannot tell which is which. Having a single big header that all the layers could read and write would destroy this transparency and make changes in the protocol of one layer visible to other layers. This is undesirable.

Question 2: What are 2 values a socket is represented by? What are 4 values a socket connection is represented by? Why do we need 4 values to represent a socket connection?

Question 3: Explain three characteristics of TCP protocol: Connection-oriented, Reliable, and Synchronous. How about UDP?

Connection-Oriented: connection-oriented describes a means of transmitting data in which the devices at the end points use a preliminary [protocol](https://searchnetworking.techtarget.com/definition/protocol) to establish an end-to-end connection before any data is sent

Reliable: TCP provides for the recovery of segments that get lost, are damaged, duplicated or received out of their correct order. TCP is described as a 'reliable' protocol because it attempts to recover from these errors.

Synchronous: TCP transmission is always asynchronous. What's synchronous or asynchronous is the behaviour of the API. A synchronous API does things while you call it: for example, send() moves data to the TCP send buffer and returns when it is done. An asynchronous API starts when you call it, executes independently after it returns to you, and calls you back or provides an interrogable handle via which completion is notified.

Question 4: What are two main issues of RPC? Explain it.

* The network between the calling process and the called process has much more complex properties than the backplane of a computer. For example, it is likely to limit message sizes and has a tendency to lose and reorder messages.
* The computers on which the calling and called processes run may have significantly different architectures and data representation formats.

Question 5: Consider a procedure *incr* with two integer parameters. The procedure adds ONE to each parameter. Now suppose that it is called with the same variable twice, for example, as *incr(i, i)*. If *i* is initially 0, what value will it have afterward if *call-by-reference* is used? How about if *copy/restore* is used?

 If call by reference is used, a pointer to *i* is passed to *incr*. It will be incre- mented two times, so the final result will be two. However, with copy/restore, *i* will be passed by value twice, each value initially 0. Both will be incre- mented, so both will now be 1. Now both will be copied back, with the second copy overwriting the first one. The final value will be 1, not 2.

Question 6: Explain why transient synchronous communication has inherent scalability problems, and how these could be solved.

The problem is the limited geographical scalability. Because synchronous communication requires that the caller is blocked until its message is received, it may take a long time before a caller can continue when the receiver is far away. The only way to solve this problem is to design the cal- ling application so that it has other useful work to do while communication takes place, effectively establishing a form of asynchronous communication.

Question 7: Can we apply the persistent asynchronous communication to RPC? Explain it.

Yes, but only on a hop-to-hop basis in which a process managing a queue passes a message to a next queue manager by means of an RPC. Effectively, the service offered by a queue manager to another is the storage of a message. The calling queue manager is offered a proxy implementation of the interface to the remote queue, possibly receiving a status indicating the success or fail- ure of each operation. In this way, even queue managers see only queues and no further communication.

Question 8: In message-oriented transient communication, why the client doesn’t need to call the *bind* primitive to bind its socket to a port?

The client declares a socket which doesn’t need explict binding to a local address, since the operating system can dynamically allocate a port when the connection is set up. It needs the IP address and port number to which a connection request is to be sent. The client is blocked until a connection has been set up successfully, after which both sides can start exchanging information through the send and receive primitives. Finally, closing a connection is symmetric when using sockets, and is established by having both the client and server call the close primitive

Question 9: In message-oriented transient communication, explain the role of the two queues (completed and incomplete connection queue) maintained by TCP for a listening socket. Explain the *backlog* parameter of the function *listen* called by a TCP server.

1. An *incomplete connection queue*, which contains an entry for each SYN that has arrived from a client for which the server is awaiting completion of the TCP three-way handshake. These sockets are in the SYN\_RCVD state.
2. A *completed connection queue*, which contains an entry for each client with whom the TCP three-way handshake has completed. These sockets are in the ESTABLISHED state.

The *backlog* argument to the listen function has historically specified the maximum value for the sum of both queues.

Question 10: What is the role of Message-Queuing System in persistent communication?

**Queues** make your data **persistent**, and reduce the errors that happen when different parts of your **system** go offline. By separating different components with **message queues**, you create more fault tolerance. If one part of the **system** is ever unreachable, the other can still continue to interact with the **queue**.

Question 11: Explain the 3 following network QoS parameters: *bit-rate*, *delay*, and *jitter*.

* Delay is an important metric in networking that measures the amount of time it takes for a bit of data to move from one endpoint to another. Delay in networking is typically on the scale of fractions of seconds, and can change based on many factors including the location of the endpoints, the size of the packet, and the amount of traffic.
* Latency is sometimes considered the time a packet takes to travel from one endpoint to another, the same as the one-way delay
* Packets transmitted continuously on the network will have differing delays, even if they choose the same path. This is inherent in a packet-switched network for two key reasons. First, packets are routed individually. Second, network devices receive packets in a queue, so constant delay pacing cannot be guaranteed. This delay inconsistency between each packet is known as jitter. It can be a considerable issue for real-time communications, including IP telephony, video conferencing, and virtual desktop infrastructure. Jitter can be caused by many factors on the network, and every network has delay-time variation.

Question 12: Concerning the method for enforcing QoS, how we can use a buffer at the receiver to reduce the jitter?

a connection with high latency won't have high jitter if the amount of latency stays constant. Conversely, jitter buffers (used to smooth out jitter) can add to overall latency by increasing playout delay, or the time between the packet's arrival and when it leaves the buffer.

Question 13: How the Forward error correction (FEC) works?

Forward error correction (FEC) is an error correction technique to detect and correct a limited number of errors in transmitted data without the need for retransmission.

In this method, the sender sends a redundant error-correcting code along with the data frame. The receiver performs necessary checks based upon the additional redundant bits. If it finds that the data is free from errors, it executes error-correcting code that generates the actual frame. It then removes the redundant bits before passing the message to the upper layers.