Computer Networking

Network Software: Connection oriented and connection less services, Service primitives, Relationship of services to protocols

Connection-Oriented and Connectionless Services

- Layers can offer two different types of service to the layers above them: **connection-oriented and connectionless.**
- **Connection-oriented service** is modeled after the telephone system. To talk to someone, you pick up the phone, dial the number, talk, and then hang up.
- Similarly, to use a connection-oriented network service, the service user **first establishes a connection**, **uses the connection**, and then **releases the connection**.
- In some cases when a connection is established, the sender, receiver, and subnet conduct a negotiation about parameters to be used, such as **maximum message size**, **quality of service required**, and **other issues.**

Connection-Oriented and Connectionless Services

- **In contrast, connectionless service** is modeled after the postal system. Each message (letter) carries the full destination address, and each one is routed through the system independent of all the others.
- Normally, when two messages are sent to the same destination, the first one sent will be the first one to arrive. However, it is possible that the first one sent can be delayed so that the second one arrives first.
- Each service can be characterized by a quality of service. Some services are reliable in the sense that they never lose data.
- Usually, a reliable service is implemented by having the receiver acknowledge the receipt of each message so the sender is sure that it arrived. The acknowledgement process introduces overhead and delays, which are often worth it but are sometimes undesirable.

Connection-Oriented and Connectionless Services

- A typical situation in which a reliable connection-oriented service is appropriate is **file transfer.**
- Reliable connection-oriented service has two minor variations: **message sequences and byte streams.**
- For some applications, the transit delays introduced by acknowledgements are unacceptable.
- Not all applications require connections.
- Unreliable (meaning not acknowledged) connectionless service is often called datagram service, in analogy with telegram service, which also does not return an acknowledgement to the sender.

Connection-Oriented and Connectionless Services

- In other situations, the convenience of not having to establish a connection to send one short message is desired, but reliability is essential.
- The acknowledged datagram service can be provided for these applications.
- Another service is the request-reply service. In this service the sender transmits a single datagram containing a request; the reply contains the answer.
- Request-reply is commonly used to implement communication in the client-server model: the client issues a request and the server responds to it.

Six different types of service

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Registered mail
	Request-reply	Database query

- The concept of using unreliable communication may be confusing at first. After all, why would anyone actually prefer unreliable communication to reliable communication?
- First of all, reliable communication (in our sense, that is, acknowledged) may not be available. For example, Ethernet does not provide reliable communication. Packets can occasionally be damaged in transit.
- Second, the delays inherent in providing a reliable service may be unacceptable, especially in real-time applications such as multimedia.

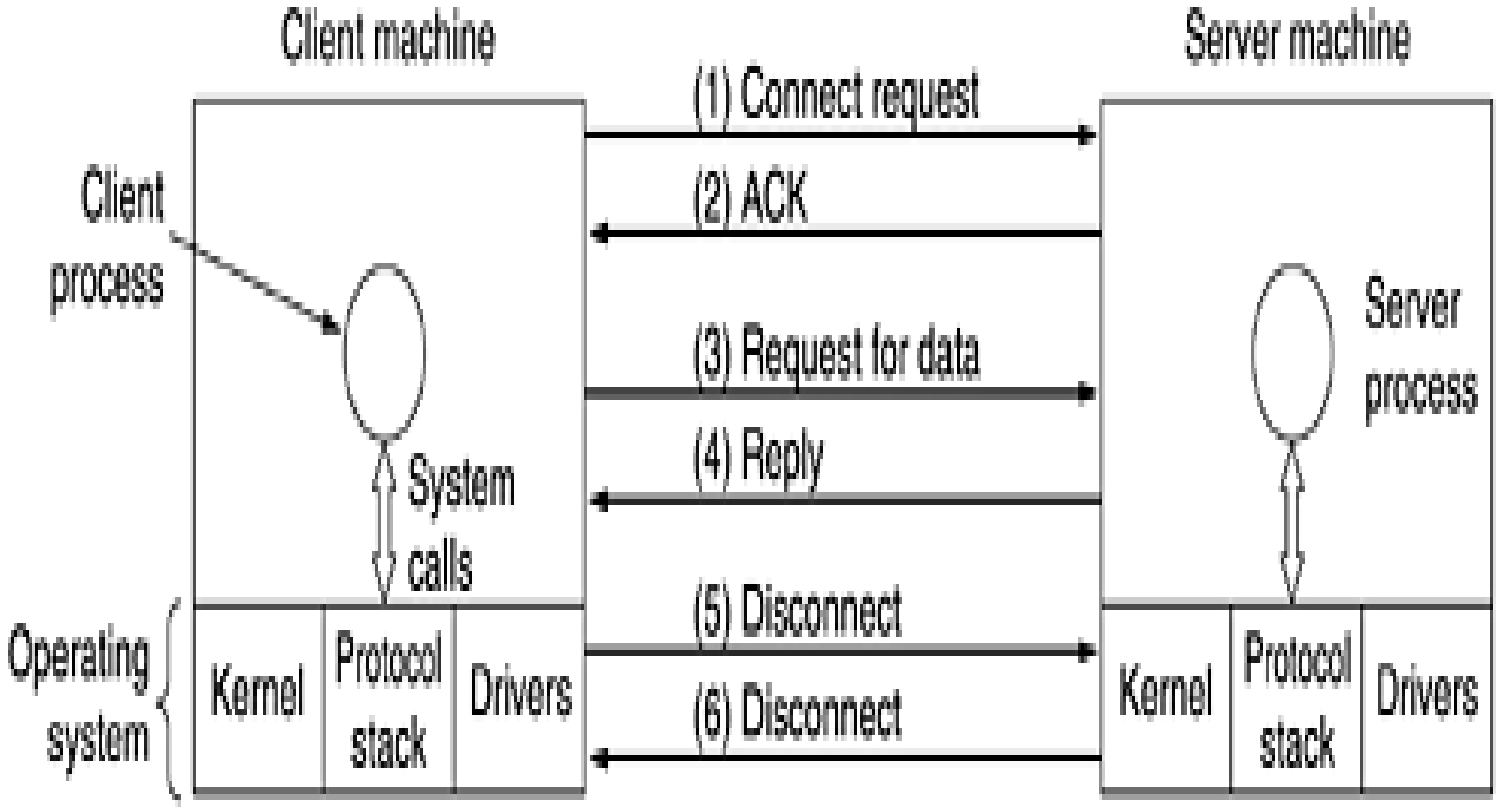
Service Primitives

- A service is formally specified by a set of primitives (operations) available to a user process to access the service.
- These primitives tell the service to perform some action or report on an action taken by a peer entity.
- The set of primitives available depends on the nature of the service being provided.
- The primitives for connection-oriented service are different from those of connectionless service.

Five service primitives for implementing a simple connection-oriented service.

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

Packets sent in a simple client-server interaction on a connection-oriented network



These primitives might be used as follows.

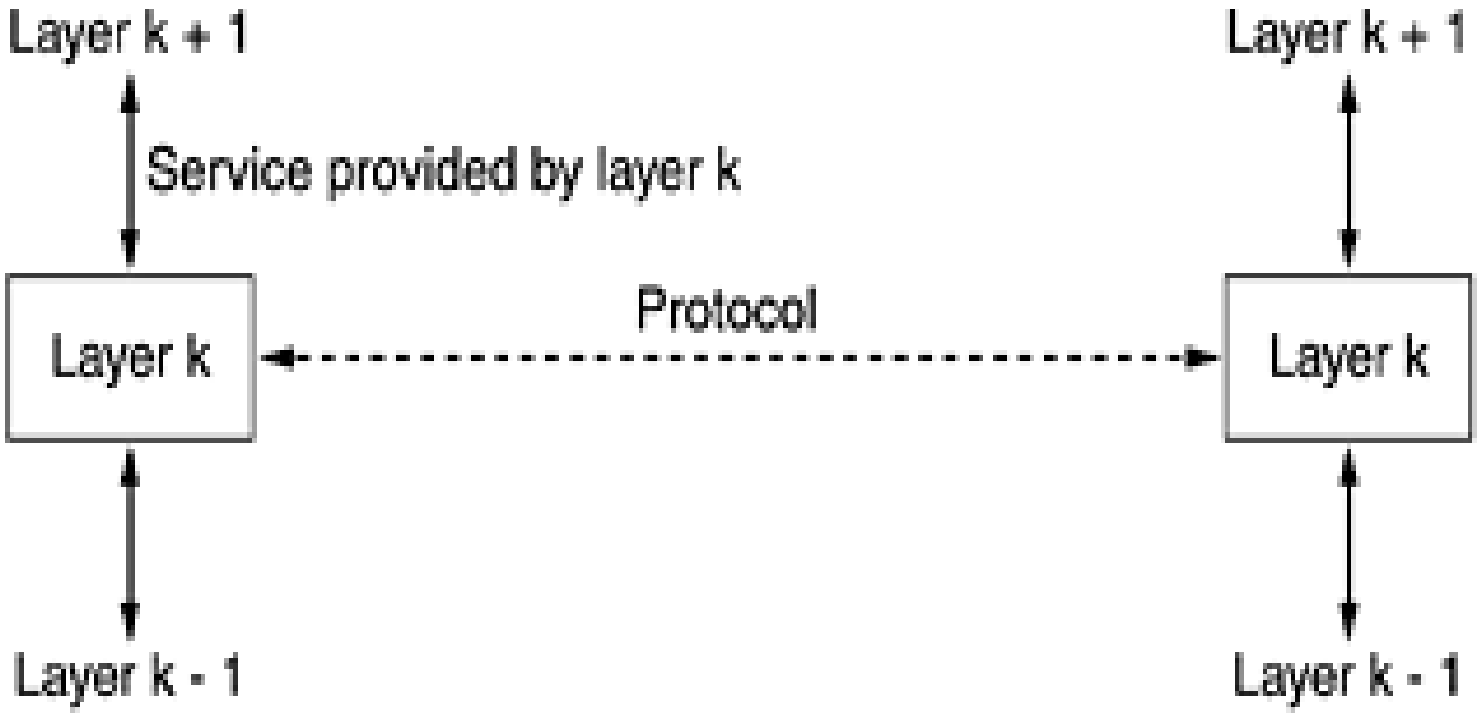
- First, the server executes LISTEN to indicate that it is prepared to accept incoming connections.
- Next, the client process executes CONNECT to establish a connection with the server.
- The CONNECT call needs to specify who to connect to, so it might have a parameter giving the server's address. The operating system then typically sends a packet to the peer asking it to connect.
- The client process is suspended until there is a response. When the packet arrives at the server, it is processed by the operating system there.
- When the system sees that the packet is requesting a connection, it checks to see if there is a listener. If so, it does two things: **unblocks the listener and sends back an acknowledgement (2).**
- The arrival of this acknowledgement then releases the client. At this point the client and server are both running and they have a connection established.
- If a connection request arrives and there is no listener, the result is undefined. In some systems the packet may be queued for a short time in anticipation of a LISTEN.

- The next step is for the server to execute RECEIVE to prepare to accept the first request. Normally, the server does this immediately upon being released from the LISTEN, before the acknowledgement can get back to the client. The RECEIVE call blocks the server.
- Then the client executes SEND to transmit its request (3) followed by the execution of RECEIVE to get the reply.
- The arrival of the request packet at the server machine unblocks the server process so it can process the request. After it has done the work, it uses SEND to return the answer to the client (4).
- The arrival of this packet unblocks the client, which can now inspect the answer. If the client has additional requests, it can make them now. If it is done, it can use DISCONNECT to terminate the connection.
- Usually, an initial DISCONNECT is a blocking call, suspending the client and sending a packet to the server saying that the connection is no longer needed (5).
- When the server gets the packet, it also issues a DISCONNECT of its own, acknowledging the client and releasing the connection. When the server's packet (6) gets back to the client machine, the client process is released and the connection is broken.

The Relationship of Services to Protocols

- A service is a set of primitives (operations) that a layer provides to the layer above it. The service defines what operations the layer is prepared to perform on behalf of its users, but it says nothing at all about how these operations are implemented.
- A service relates to an interface between two layers, with the lower layer being the service provider and the upper layer being the service user.
- A protocol, in contrast, is a set of rules governing the format and meaning of the packets, or messages that are exchanged by the peer entities within a layer.
- Entities use protocols to implement their service definitions. They are free to change their protocols at will, provided they do not change the service visible to their users.
- In this way, the service and the protocol are completely decoupled.

The relationship between a service and a protocol



QUIZ TEST