31341

Assignment 4

Problem Statement: Write a program using TCP socket for wired network for the following
a. Say Hello to Each other
b. File transfer
c. Calculator (Arithmetic)
d. Calculator (Trigonometry)
Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Objective:
Setup connection TCP between two nodes
a. Say Hello to Each other
b. File transfer
c. Calculator (Arithmetic)
d. Calculator (Trigonometry)

Learning Outcome: Students will be able to
- Demonstrate working of TCP sockets
- Perform the above operations over TCP sockets

Requirements:
- Open source linux based OS
- Eclipse IDE or Python interpreter

Theory
TCP
TCP is connection-oriented, and a connection between client and server is established before data can be sent. The server must be listening (passive open) for connection requests from clients before a connection is established. Three-way handshake (active open), re-transmission, and error-detection adds to reliability but lengthens latency. TCP employs

network congestion avoidance. However, there are vulnerabilities to TCP including denial of service, connection hijacking, TCP veto, and reset attack. For network security, monitoring, and debugging, TCP traffic can be intercepted and logged with a packet sniffer.

TCP Segment structure

A TCP segment consists of a segment header and a data section. The segment header contains 10 mandatory fields, and an optional extension field (Options, pink background in table). The data section follows the header and is the payload data carried for the application. The length of the data section is not specified in the segment header; It can be calculated by subtracting the com bined length of the segment header and IP header from the total IP datagram length specified in the IP header.

Source port (16 bits): Identifies the sending port.

Destination port (16 bits): Identifies the receiving port.

Sequence num ber (32 bits)
Has a dual role:
· If the SYN flag is set (1), then this is the initial sequence num ber. The sequence number of the actual first data byte and the acknowledged number in the corresponding ACK are then this sequence number plus 1.
· If the SYN flag is clear (0), then this is the accumulated sequence number of the first data byte of this segment for the current session.

Acknowledgment number (32 bits)
If the ACK flag is set then the value of this field is the next sequence number that the sender of the ACK is expecting. This acknowledges receipt of all prior bytes (if any). The first ACK sent by each end acknowledges the other end's initial sequence num ber itself, but no data.

Data offset (4

bits)
Specifies the size of the TCP header in 32-bit words. The minimum size header is 5 words and the maximum is 15 words thus giving the minimum size of 20 bytes and maximum of 60 bytes, allowing for up to 40 bytes of options in the header. This field gets its name from the fact that it is also the offset from the start of the TCP segment to the actual data.

Reserved (3 bits)
For future use and should be set to zero.

Flags (9 bits)
Contains 9 1-bit flags (control bits) as follows:
• NS (1 bit): ECN-nonce - concealment protection
• CWR (1 bit): Congestion window reduced (CWR) flag is set by the sending host to indicate that it received a TCP segment with the ECE flag set and responded in a congestion control mechanism.
• ECE (1 bit): ECN-Echo has a dual role, depending on the value of the SYN flag. It indicates:
• If the SYN flag is set (1), that the TCP peer is ECN capable.
• If the SYN flag is clear (0), that a packet with Congestion Experienced flag set (ECN=11) in the IP header was received during normal transmission. This serves as an indication of network congestion (or impending congestion) to the TCP sender.
• URG (1 bit): Indicates that the Urgent pointer field is significant
• ACK (1 bit): Indicates that the Acknowledgment field is significant. All packets after the initial SYN packet sent by the client should have this flag set.
• PSH (1 bit): Push function. Asks to push the buffered data to the receiving application.
• RST (1 bit): Reset the connection
• SYN (1 bit): Synchronize sequence numbers. Only the first packet sent from each end should have this flag set. Some other flags and fields change meaning based on this flag, and some are only valid when it is set, and others when it is clear.
• FIN (1 bit): Last packet from sender

Window size (16

bits)
The size of the receive window, which specifies the number of window size units that the sender of this segment is currently willing to receive.

Checksum (16 bits)
The 16-bit checksum field is used for error-checking of the TCP header, the payload and an IP pseudo-header. The pseudo-header consists of the source IP address, the destination IP address, the protocol number for the TCP protocol (6) and the length of the TCP headers and payload (in bytes).

Urgent pointer (16 bits)
If the URG flag is set, then this 16-bit field is an offset from the sequence number indicating the last urgent data byte.

Options (Variable 0–320 bits, in units of 32 bits)

TCP provides the concept of a connection. A process creates a TCP socket by calling the socket() function with the parameters PF_INET or PF_INET6 and SOCK_STREAM.
Server
Setting up a simple TCP server involves the following steps:
1. Creating a TCP socket, with a call to socket().

2. Binding the socket to the listen port, with a call to bind(). Before calling bind(), sockaddr_in structure must be declared, and it and the sin_family (AF_INET or AF_INET6) must be cleared, and fill its sin_port (the listening port, in network byte order) fields. Converting a short int to network byte order can be done by calling the function htons() (host to network short).

3. Preparing the socket to listen for connections (making it a listening socket), with a call to

listen().

4. Accepting incoming connections, via a call to accept(). This blocks until an incoming connection is received, and then returns a socket descriptor for the accepted connection. The initial descriptor remains a listening descriptor, and accept() can be called again at any time with this socket, until it is closed.

5. Communicating with the remote host, which can be done through send() and recv().

6. Eventually closing each socket that was opened, once it is no longer needed, using close(). Note that if there were any calls to fork(), each process must close the sockets it knew about (the kernel keeps track of how many processes have a descriptor open), and two processes should not use the same socket at once.

Client

Setting up a TCP client involves the following steps:

1. Creating a TCP socket, with a call to socket().

2. Connecting to the server with the use of connect, passing a sockaddr_in structure with the sin_family set to AF_INET or AF_INET6, sin_port set to the port the endpoint is listening (in network byte order), and sin_addr set to the IPv4 or IPv6address of the listening server (also in network byte order.)

3. Communicating with the server by send()ing and recv()ing.Terminating the connection and cleaning up with a call to close(). Again, if there were any calls to fork(), each process must close() the socket.

Conclusion:

Thus we have successfully implemented the socket programming for TCP using C.