Short(er) Answer questions

1. Calculate the total time required to transfer a 1000-kB file in the following cases, assuming an RTT of 100ms, a packet size of 1kB data, and an initial 2xRTT of "handshaking" before the data are sent out.

   -The bandwidth is 1.5 Mbps, and data packets can be sent continuously. (5 points)

Ans:
initial 2*RTT handshaking: 2 * 100ms = 200ms
transmit time: 1000kB / 1.5Mbps = 8192Mb / 1.5Mbps = 5461.33 ms
propagation: RTT/2 = 50ms
total time: 5461.33 + 200 + 50 = 5711.33ms = 5.71s

   -The bandwidth is 1.5 Mbps, but after we finish sending each data packet, we must wait one RTT before sending the next. (5 points)

Ans:
number of packets: 1000 / 1 = 1000
total = 5711.33 + (1000-1) * 100 = 105611.33 ms = 105s

   -The bandwidth is "infinite", meaning that we take the transmit time to be zero, and up to 20 packets can be sent per RTT (5 points)

Ans:
initial 2*RTT handshaking: 2 * 100ms = 200ms
packets number = 1000 kB / 1 kB = 1000
number of transmit = 1000 / 20 = 50
time of transmit = 50 * 100 = 5000ms
total = 5000 + 200 = 5200 ms = 5.2s

2. What are the major operations (specifically the functions) performed by the client part of a socket interface? Describe each one of them. and what they do. (5 points)

Ans:
1.int socket(int domain, int type, int protocol);

create a socket

2.int connect(int socket, struct sockaddr *address, int addr_len);
this is used established connection between client and server.

3.int send(int socket, char *message, int msg_len, int flags);
int recv(int socket, char *buffer, int buf_len, int flags);
Send operation sends the given message over the specified socket, while the Recv operation receives a message from the specified socket into the given buffer.

4.close(): close socket and release resource.

3. What are the major operations (specifically the functions) performed by the server part of a socket interface? Describe each one of them, and what they do. (5 points)

Ans:
1.int socket(int domain, int type, int protocol);
create a socket

2.int bind(int socket, struct sockaddr *address, int addr_len);
The bind operation, binds the newly created socket to the specified address.

3.int listen(int socket, int backlog);
4.int accept(int socket, struct sockaddr *address, int *addr_len);
The listen operation then defines how many connections can be pending on the specified socket. Finally, the accept operation carries out the passive open.

5.int send(int socket, char *message, int msg_len, int flags);
6.int recv(int socket, char *buffer, int buf_len, int flags);
Send operation sends the given message over the specified socket, while the Recv operation receives a message from the specified socket into the given buffer.

7.close(): close socket and release resource.
(https://book.systemsapproach.org/foundation/software.html)

4. Look at the socket code. There's a struct in there, a struct of type sock_addr_in with a variable name of sin. Use the Linux man pages or https://learning.oreilly.com/library/view/the-linux-programming/9781593272203/xhtml/ch56.xhtml to find out the members/fields of this struct. (5 points)

```
struct sockaddr_in {            /* IPv4 socket address */
   sa_family_t sin_family;      /* Address family (AF_INET) */
   in_port_t sin_port;          /* Port number */
   struct in_addr sin_addr;     /* IPv4 address */
   unsigned char __pad[X];      /* Pad to size of 'sockaddr'
                                   structure (16 bytes) */
};
```

5. The listen() function takes a parameter called backlog, an integer. Describe what this parameter does. You can test the client & server code that you have with different backlog values, and you can also read about it in both Computer Networking: A Systems Approach section 1.4Links to an external site. and Chapter 56 of The Linux Programming InterfaceLinks to an external site. (and both are searchable). (5 points)

Ans:
backlog variable is the max length of queue for pending connections. If the queue is full, the server will refuse new connections.

Long Response Questions - answers to these questions will be at least a paragraph long.

1. What are port numbers used for in the socket code in Section 1.4 of Computer Networking: A Systems Approach? Why do we need port numbers when receiving data from the internet on our computers? Explain the function that port numbers perform in the code. (5 points)

Port numbers used in the socket code in Section 1.4 of Computer Networking: A Systems Approach is 5432.

Port numbers are needed when receiving data from the internet because it can direct the data to the specific application or process. Without port numbers, data cannot be directed to correct application.

In the client code, port numbers specify the endpoint of communication between client and server. Server bind and listen at port 5432, and client send request through port 5432, so

that server can accept the connect request, and port number are served as identifier between client and server.

2. The tcp_client.c code has a while loop in lines 54-57. What is the purpose of this while loop? Under what condition would it terminate, and why or how does it terminate? What are the purposes of buf and send in this loop? (10 points)

Ans:
Purpose of while loop is to continuously read the input from user using stdin.

It terminates under condition when fgets() function returns NULL, a NULL return value indicates error or end-of-file condition. It is because there is no more input to read from the user.

buf is a char array that stores user input with max length 256, send function sends the given message over the socket.

3. The tcp_server.c code has a pair of nested while loop from lines 39 to 46. Describe what each line of code from 39 to 46 is doing: if it's a loop, describe what will cause the loop to stop looping. If it's a conditional, describe what the condition is; make sure that your answers are written in terms of what the tcp_server and tcp_client are doing (so don't just describe that x is an integer; explain what the integer means). If there is a function that returns a value, explain the function and its parameters. Ultimately what you want to be explaining here is how the server runs, when it stops, and how it deals with new connections. Include how many clients can be connected at one time, how many clients can be waiting to connect, and what code controls this feature. Make sure to experiment with the client and server code to make sure that you have the correct answers here. (20 points)

Answer:
line 39 while(1):
creates an infinite loop for waiting client connection

line 40 if ((new_s = accept(s, (struct sockaddr *)&sin, &addr_len)) < 0):
accept function waiting for new connection from client and accept, parameter: s is current socket file descriptor, (struct sockaddr *)&sin: we get address of input from client, which is a struct sockaddr_in, and cast it into pointer of sockaddr struct, since accept() function need sockaddr pointer as second parameter. &addr_len is a pointer to address of addr_len, contain the amount of space pointed to by address from input client. The return value is a

new socket file descriptor and we assign it to new_s. If the condition check that if new socket file descriptor is less than 0 which if client fails to accept the connection, because accept() returns -1 on error.

line 41 perror("simplex-talk: accept"):
if error occurs, call perror(), which finds the error message corresponding to the current value of the global variable errno nd writes it

line 42 exit(1):
if error occurs, quit program

line 44 while (buf_len = recv(new_s, buf, sizeof(buf), 0)):
recv function receives a message from the socket into the given buffer, take in three argument socket file descriptor new_s, buf is a char array that stores user input with max length 256, sizeof function return the length of buf array. recv function return 0 or negative value, the return value 0 means the peer has closed its half side of the connection. While loop continues when buf_len is 0, meaning that the server has closed the connection.

line 45 fputs(buf, stdout):
In server side, this line prints the received data in buf array to standard output

line 46 close(new_s):
close(): close function socket file descriptor new_s and release resource