I.a.1 The listening socket is bound to a specific address. What address is this? (Give

both the symbolic name used in the code, and the corresponding IPv4 address in numeric or

dotted notation).

symbolic name INADDR\_ANY

numeric name 127.0.0.1

I.a.2 In the code, there is a call to recv() as follows:

ret = recv( cd.sock, cd.buffer, kTransferBufferSize, 0 );

The return value ret will be one of the following:

(a) ret = -1

(b) ret = 0

(c) 0 < ret < kTransferBufferSize

(d) ret = kTransferBufferSize

Describe the implications of each case! Additionally, why is cd.buffer (see ConnectionData declaration) defined to be of size kTransferBufferSize+1 rather than just plain kTransferBufferSize?

1. Error in recv
2. Shutdown successfully
3. The data in the buffer has been copied to ret and the value will be between 0 and 64
4. All the data in the buffer filled to ret while the remaining have be discarded

Since kTransferBufferSize starts from 0, 1is added to make space for terminated the processes failure. And it’s intelligent way to track if the buffer is full and keep track the maximum size of the buffer.

I.a.3 Sending is performed using the send() method as follows:

ret = send( cd.sock,

cd.buffer+cd.bufferOffset,

cd.bufferSize-cd.bufferOffset,

MSG\_NOSIGNAL

);

How does the send() method indicate that the connection in question has been closed/reset?

How does MSG\_NOSIGNAL relate to this (on linux machines)?

1. After trigger **MSG\_CONFIRM** Then wait for a subsequent recv() call on that socket to return 0 (it emptys the buffer and change the connection to recive)
2. MSG\_NOSIGNAL: do not allow for kernel to generate **SIGPIPE** signal if the peer on a stream-oriented socket has closed or reset the connection.

I.a.4 Discuss the reasons for this behaviour with your partner. Why are these two

strategies used?

Also, quickly look through the error codes (values of errno) possible after accept(), send(),

and recv() (check the man-pages!). Under which conditions attempting to continue execution

might be unreasonable?

The first approach is to ensure that there is a connection. If it is unable to accept or bind, then exit

The second approach, is to drop requests from clients that cannot be established and start processing a new connection from a client, in waiting

I.b.1 Discuss with your partner: How is the program notified that a connection

attempt has failed or succeeded?

Hint: the process is described in the course book!

Bothe condition will well occur, when there is no connection errors (connection error is zero) it notifies the client that the connection is successful other wise it will check number of error with number of connection and if it’s equaled the number of client it mean all connection are in error and has field. Hens if one client field it well return error for this client but the server will continue with the others.

I.c.1 Try to send messages with each of the clients. Describe the results – do you

receive a response immediately?

Check with netstat and document the status of the connection from each client.

Once connections between the server and the clients. The server only responds to one client at a time. Messages from other clients will be discarded

Once the socket connection of the first client is killed, the server starts processing the request of the new client

I.c.2 When you disconnected the first client, what happened? Explain why.

After an orderly shutdown of the connection with client one, the socket was reestablished for client 2 and the communication was restored with the same socket

I.c.3 Measure the round trip time when the client and server are running on the

same machine. Also measure the round trip time when they are on different machines.

Can you observe any differences? Write down the times. (Note: take the average of a few

(> 5) attempts.)

I.c.4 Measure the round trip times for two concurrently connected simple clients

(similar to exercise I.c.1 ).

Discuss with your partner: What is the largest factor in the measured round trip time of the

second client?

I.d.1 Run the above command (make sure that the server is still running), and note

the results.

I.d.2 Take note of the timing results. You will want to compare them to results in

the next Lab/Exercise.

(You don’t have to hand in the results, though.)

I.d.3 How long did it take for the connection attempts to time out?

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Sent** | **Round Trip Time (milliseconds – ms)**  **Client & Server on Same Machine** | **Round Trip Time (milliseconds – ms)**  **Client & Server on Different Machine** | **Remarks** |
| This is a test sent to the server – 33 bytes | 0.084463 | 5.936388 | It was faster when client and server are on the same machine |
| I am enjoying this lab work – 27 bytes | 0.100951 | 9.161370 |  |
| It is really nice connecting to chalmers remote server – 54 bytes | 0.083655 | 4.706442 |  |
| This is a test sent to the server – 33 bytes | 0.092891 | 11.162449 |  |
| Test Data – 9 bytes | 0.108161 | 10.770205 |  |
| I am enjoying this lab work – 27 bytes | 0.061767 | 9.626789 |  |
| It will be a nice experience studying in Chalmers Sweden. Looking forward to the experiencing the culture – 105 bytes | 0.071855 | 29.402143 |  |

Q 2:

Your answer:

The symbolic name is INADDR\_ANY

The IP address is 0.0.0.0

Q 3: What if the client sent 65 bytes?

1. ret = -1

This means that there is an error in recv()

2. ret= 0

This means that the shutdown process was orderly done

3. 0 < ret < kTransferBufferSize

This means that the data received in the buffer, measured in bytes, is

between the range of between 0 and 64 bytes

4. ret = kTransferBufferSize

This means that 64bytes were actually received based on the recv()

Function

IF client sent 65 bytes, it will accept the 64 bytes, reply, then reset the buffer and be ready to accept the remaining bytes.

Since kTransferBufferSize starts from 0, 1is added to make space for

track the end of the buffer. It is an intelligent way to track if the buffer

is full and keep track the maximum size of the buffer. This is like

creating a circular memory.

Q 4: "How does the send() method indicate that the connection in question has been closed/reset?"

The **send** () indicates a closed/reset connection by clearing/zeroing the buffer and changing the connection state to receiving., or simply by indicating zero value in the buffer.

After rec() send last ACK then ask to FIN the connection.

The send() method is used to empty the buffer and change the

connection state to a new receiving state

MSG\_NOSIGNAL: This does not allow for Linux kernel to generate

SIGPIPE signal if the peer on a stream-oriented socket has closed or

reset the connection. SIGPIPE is a signal sent to a process when it

attempts to write a pipe without a process connected to the other end

Q 5: "Under which conditions attempting to continue executionmight be unreasonable?"

Your answer:

The first approach is an exit method based on the status of the

connection. If it is unable to accept or bind, or create a connection,

then it simply exits.

The second approach, is to drop requests from clients that cannot be

established and start a new process to connect newer connections

from another client, that maybe in waiting. Since it is an iterative

server, it processes on connection at a time.

It will be unreasonable to continue execution under the following error conditions:

* Socket is nonblocking and there’s no connection to accept.
* Terminated connections.
* No listening socket connections, or address size is invalid/negative.
* The per-process limit on the number of open file descriptors has been reached.
* The system-wide limit on the total number of open files has been reached.
* Not enough free memory. This often means that the memory allocation is limited by the socket buffer limits, not by the system memory.

Q 6:

Your answer:

The program decides on what to do based on the connection errors.

If the number of errors, as coded, is greater than zero, it goes ahead

to compare this with the number of clients. Then, if the number of

connections is equal to the number of clients, then it returns that all

connection are in error and has failed.

If the above does not hold, then it returns the total number of

successful connections on the server. Even if one client connection

fails, it will return error for this client but the server will continue with

the other connections.

Client response:

haitham@remote11:~/lab1.2$ ./client-simple remote11.chalmers.se 31336

Input> hello

Sending string `hello' (5 bytes)

Response = `hello'

- response does match original query

- round trip time is 0.065681 ms

Single client: server response:

haitham@remote11:~/lab1.2$ ./server 31336

Attempting to bind to port 31336

Socket is bound to 0.0.0.0 31336

Connection from 129.16.29.50:39782 -> socket 4

Multi-client sent:

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 10 10

Simulating 10 clients.

Establishing 10 connections...

successfully initiated 10 connection attempts!

Connect timing results for 10 successful connections

- min time: 0.215181 ms

- max time: 0.462316 ms

- average time: 0.297343 ms

(0 connections failed!)

Roundtrip timing results for 10 connections for 10 round trips

- min time: 0.279095 ms

- max time: 2.349003 ms

- average time: 1.340303 ms

Client-muli to server : server response. For 10u 10m

Attempting to bind to port 31336

Socket is bound to 0.0.0.0 31336

Connection from 129.16.29.50:39760 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39762 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39764 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39766 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39768 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39770 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39772 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39774 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39776 -> socket 4

socket 4 - orderly shutdown

Connection from 129.16.29.50:39778 -> socket 4

socket 4 - orderly shutdown

Client- multi : client side

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 100 10000

Simulating 100 clients.

Establishing 100 connections...

successfully initiated 100 connection attempts!

- conn 92 : error in recv() : Connection reset by peer

- conn 47 : error in recv() : Connection reset by peer

- conn 48 : error in recv() : Connection reset by peer

- conn 49 : error in recv() : Connection reset by peer

- conn 50 : error in recv() : Connection reset by peer

- conn 51 : error in recv() : Connection reset by peer

- conn 52 : error in recv() : Connection reset by peer

- conn 53 : error in recv() : Connection reset by peer

- conn 54 : error in recv() : Connection reset by peer

- conn 55 : error in recv() : Connection reset by peer

- conn 56 : error in recv() : Connection reset by peer

- conn 57 : error in recv() : Connection reset by peer

- conn 58 : error in recv() : Connection reset by peer

- conn 59 : error in recv() : Connection reset by peer

- conn 60 : error in recv() : Connection reset by peer

- conn 61 : error in recv() : Connection reset by peer

- conn 62 : error in recv() : Connection reset by peer

- conn 63 : error in recv() : Connection reset by peer

- conn 64 : error in recv() : Connection reset by peer

- conn 65 : error in recv() : Connection reset by peer

- conn 66 : error in recv() : Connection reset by peer

- conn 67 : error in recv() : Connection reset by peer

- conn 68 : error in recv() : Connection reset by peer

- conn 69 : error in recv() : Connection reset by peer

- conn 70 : error in recv() : Connection reset by peer

- conn 71 : error in recv() : Connection reset by peer

- conn 72 : error in recv() : Connection reset by peer

- conn 73 : error in recv() : Connection reset by peer

- conn 74 : error in recv() : Connection reset by peer

- conn 75 : error in recv() : Connection reset by peer

- conn 15 : error in recv() : Connection reset by peer

- conn 16 : error in recv() : Connection reset by peer

- conn 17 : error in recv() : Connection reset by peer

- conn 18 : error in recv() : Connection reset by peer

- conn 19 : error in recv() : Connection reset by peer

- conn 20 : error in recv() : Connection reset by peer

- conn 21 : error in recv() : Connection reset by peer

- conn 22 : error in recv() : Connection reset by peer

- conn 23 : error in recv() : Connection reset by peer

- conn 24 : error in recv() : Connection reset by peer

- conn 25 : error in recv() : Connection reset by peer

- conn 26 : error in recv() : Connection reset by peer

- conn 27 : error in recv() : Connection reset by peer

- conn 28 : error in recv() : Connection reset by peer

- conn 29 : error in recv() : Connection reset by peer

Connect timing results for 100 successful connections

- min time: 1.427047 ms

- max time: 1004.916701 ms

- average time: 904.245411 ms

(0 connections failed!)

Roundtrip timing results for 55 connections for 10000 round trips

- min time: 706.457603 ms

- max time: 59543.044363 ms

- average time: 25366.642656 ms

Q7:

Your answer:

Once we had established connections to different clients, we sent

messages from the first and the second. The server responded to

immediately only the first one and seem to discard the messages

from the second. We also got a message "response does not match

original query" a times.

The netstat command shows the connection status of the sockets

and their corresponding process identifiers (PID)

Q8:

Your answer:

After an orderly shutdown of the connection with the first client, the

session was re-established for the second client, and the messages

being sent from this client now started getting responses from the

server. We take it that the iterative server was only responding to one

client at a time. When we ran the netstat command, the PID for the

first client was nom longer seen on the list. We can infer that this is

simply how the iterative server works

Q 9:

Your answer:

|  |  |  |
| --- | --- | --- |
| **Size of Data Sent (Bytes)** | **RTT (ms)**  **Client & Server are on the same machine** | **RTT (ms)**  **Client & Server on Different Machine** |
| 10 | 0.108161 | 10.770205 |
| 26 | 0.100951 | 9.161370 |
| 27 | 0.061767 | 9.626789 |
| 30 | 0.084463 | 5.936388 |
| 33 | 0.092891 | 11.162449 |
| 54 | 0.083655 | 4.706442 |
| 104 | 0.061767 | 9.626789 |
| Average value | 0.08480786 | 8.71291886 |

Yes, we observed clear differences between the scenarios. The response was far better, lower in value, when the server and client were running on the same machine. The response time was higher when the client connected remotely to the server.

Q 10: Based on your explanation, what do you think "is the largest factor in the measured round-trip time of the second client?"

To handle concurrent connections the iterative server program has to be modified to accept simultaneous connections from clients. The iterative server takes one connection at a time, and so we could not initiate two concurrent sessions.

Intuitively, for concurrent sessions, the server response time will play a major role in creating some difference in the round-trip time. The number of hops, distance, traffic level and transmission channel/medium will be the same for both clients at the same time

The largest factor in the RTT measured values is the amount of the message that has been sent to the server.

Q11: Provide some numbers to sustain your statement,

We observed that as the number of clients was being increased, the minimal and maximal times, also increased as well.

Also, the RTT values increased as the number of clients increase. We therefore see a direct proportionality between the number of clients and the minimal/maximal/RTT values.

Last time, we observed a few errors that seem to also increase with thenumber of clients. For example, we got:

conn 23 : error in recv() : Connection reset by peer

conn 25 : error in recv() : Connection reset by peer

But this Time, we observed no error, unless if the number of client and number of message increased so much.

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 7 255

Simulating 7 clients.

Establishing 7 connections...

successfully initiated 7 connection attempts!

Connect timing results for 7 successful connections

- min time: 0.173636 ms

- max time: 0.427089 ms

- average time: 0.275111 ms

(0 connections failed!)

Roundtrip timing results for 7 connections for 255 round trips

- min time: 9.985360 ms

- max time: 63.977426 ms

- average time: 37.188953 ms

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 10 255

Simulating 10 clients.

Establishing 10 connections...

successfully initiated 10 connection attempts!

Connect timing results for 10 successful connections

- min time: 0.214704 ms

- max time: 0.478328 ms

- average time: 0.324034 ms

(0 connections failed!)

Roundtrip timing results for 10 connections for 255 round trips

- min time: 10.629230 ms

- max time: 92.910074 ms

- average time: 52.440092 ms

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 15 255

Simulating 15 clients.

Establishing 15 connections...

successfully initiated 15 connection attempts!

Connect timing results for 15 successful connections

- min time: 0.265705 ms

- max time: 1031.365713 ms

- average time: 344.020323 ms

(0 connections failed!)

Roundtrip timing results for 15 connections for 255 round trips

- min time: 10.183681 ms

- max time: 103.961766 ms

- average time: 46.855492 ms

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 30 255

Simulating 30 clients.

Establishing 30 connections...

successfully initiated 30 connection attempts!

Connect timing results for 30 successful connections

- min time: 0.469887 ms

- max time: 1019.959777 ms

- average time: 680.125051 ms

(0 connections failed!)

Roundtrip timing results for 30 connections for 255 round trips

- min time: 11.067528 ms

- max time: 248.381284 ms

- average time: 114.908063 ms

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 50 255

Simulating 50 clients.

Establishing 50 connections...

successfully initiated 50 connection attempts!

Connect timing results for 50 successful connections

- min time: 0.643474 ms

- max time: 1011.891136 ms

- average time: 809.576258 ms

(0 connections failed!)

Roundtrip timing results for 50 connections for 255 round trips

- min time: 12.587867 ms

- max time: 957.579719 ms

- average time: 346.776135 ms

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 100 255

Simulating 100 clients.

Establishing 100 connections...

successfully initiated 100 connection attempts!

Connect timing results for 100 successful connections

- min time: 1.384085 ms

- max time: 1004.313830 ms

- average time: 903.404359 ms

(0 connections failed!)

Roundtrip timing results for 100 connections for 255 round trips

- min time: 10.155905 ms

- max time: 26756.988687 ms

- average time: 5381.332484 ms

No error

Q12: Perform the simple DoS attack proposed and count the time until connections start failing.

It took about 1:10 minuts

haitham@remote11:~/lab1.2$ ./client-multi remote11.chalmers.se 31336 75 75

Simulating 75 clients.

Establishing 75 connections...

successfully initiated 75 connection attempts!

- conn 0 : async connect() error: Connection timed out

- conn 1 : async connect() error: Connection timed out

- conn 2 : async connect() error: Connection timed out

- conn 3 : async connect() error: Connection timed out

- conn 4 : async connect() error: Connection timed out

- conn 5 : async connect() error: Connection timed out

- conn 6 : async connect() error: Connection timed out

- conn 7 : async connect() error: Connection timed out

- conn 8 : async connect() error: Connection timed out

- conn 9 : async connect() error: Connection timed out

- conn 10 : async connect() error: Connection timed out

- conn 11 : async connect() error: Connection timed out

- conn 12 : async connect() error: Connection timed out

- conn 13 : async connect() error: Connection timed out

- conn 14 : async connect() error: Connection timed out

- conn 15 : async connect() error: Connection timed out

- conn 16 : async connect() error: Connection timed out

- conn 17 : async connect() error: Connection timed out

- conn 18 : async connect() error: Connection timed out

- conn 19 : async connect() error: Connection timed out

- conn 20 : async connect() error: Connection timed out

- conn 21 : async connect() error: Connection timed out

- conn 22 : async connect() error: Connection timed out

- conn 23 : async connect() error: Connection timed out

- conn 24 : async connect() error: Connection timed out

- conn 25 : async connect() error: Connection timed out

- conn 26 : async connect() error: Connection timed out

- conn 27 : async connect() error: Connection timed out

- conn 28 : async connect() error: Connection timed out

- conn 29 : async connect() error: Connection timed out

- conn 30 : async connect() error: Connection timed out

- conn 31 : async connect() error: Connection timed out

- conn 32 : async connect() error: Connection timed out

- conn 33 : async connect() error: Connection timed out

- conn 34 : async connect() error: Connection timed out

- conn 35 : async connect() error: Connection timed out

- conn 36 : async connect() error: Connection timed out

- conn 37 : async connect() error: Connection timed out

- conn 38 : async connect() error: Connection timed out

- conn 39 : async connect() error: Connection timed out

- conn 40 : async connect() error: Connection timed out

- conn 41 : async connect() error: Connection timed out

- conn 42 : async connect() error: Connection timed out

- conn 43 : async connect() error: Connection timed out

- conn 44 : async connect() error: Connection timed out

- conn 45 : async connect() error: Connection timed out

- conn 46 : async connect() error: Connection timed out

- conn 47 : async connect() error: Connection timed out

- conn 48 : async connect() error: Connection timed out

- conn 49 : async connect() error: Connection timed out

- conn 50 : async connect() error: Connection timed out

- conn 51 : async connect() error: Connection timed out

- conn 52 : async connect() error: Connection timed out

- conn 53 : async connect() error: Connection timed out

- conn 54 : async connect() error: Connection timed out

- conn 55 : async connect() error: Connection timed out

- conn 56 : async connect() error: Connection timed out

- conn 57 : async connect() error: Connection timed out

- conn 58 : async connect() error: Connection timed out

- conn 59 : async connect() error: Connection timed out

- conn 60 : async connect() error: Connection timed out

- conn 61 : async connect() error: Connection timed out

- conn 62 : async connect() error: Connection timed out

- conn 63 : async connect() error: Connection timed out

- conn 64 : async connect() error: Connection timed out

- conn 65 : async connect() error: Connection timed out

- conn 66 : async connect() error: Connection timed out

- conn 67 : async connect() error: Connection timed out

- conn 68 : async connect() error: Connection timed out

- conn 69 : async connect() error: Connection timed out

- conn 70 : async connect() error: Connection timed out

- conn 71 : async connect() error: Connection timed out

- conn 72 : async connect() error: Connection timed out

- conn 73 : async connect() error: Connection timed out

- conn 74 : async connect() error: Connection timed out

Connect timing results for 0 successful connections

- min time: inf ms

- max time: 0.000000 ms

- average time: -nan ms

(75 connections failed!)

Roundtrip timing results for 0 connections for 75 round trips

- min time: inf ms

- max time: 0.000000 ms

- average time: -nan ms