1

00:00:05,899 --> 00:00:10,679

welcome to QB64 report

I'm your host Fellippe Heitor and in today's

2

00:00:10,679 --> 00:00:15,179

episode Dave and I will talk to Luke one

of the developers maintaining QB64

3

00:00:15,179 --> 00:00:19,830

let's hear about tcp/ip communications

how to create a simple server program

4

00:00:19,830 --> 00:00:24,480

and a simple client program to connect

to it both locally and over the Internet

5

00:00:24,480 --> 00:00:31,050

how exciting is that without further ado

here's Luke hey man you're on I believe

6

00:00:31,050 --> 00:00:35,219

that everybody else I'd like to talk a

little bit about networking in QB64

7

00:00:35,219 --> 00:00:40,110

specifically we're talking about tcp/ip

connections which although you may not

8

00:00:40,110 --> 00:00:44,640

have heard of them are pretty much

everywhere in today's world so they're

9

00:00:44,640 --> 00:00:47,460

what underpins everything on the

Internet

10

00:00:47,460 --> 00:00:52,920

for the most part aside from podcasts

such as this and they're used even

11

00:00:52,920 --> 00:00:57,090

locally on computers for programs to

talk to each other they are absolutely

12

00:00:57,090 --> 00:01:02,280

essential if you want to do any kind of

communication between computers the

13

00:01:02,280 --> 00:01:09,000

basics begin with the idea of a client

and a server so you of course are

14

00:01:09,000 --> 00:01:14,250

familiar with the idea of web browsers

talking to web sites the web site is

15

00:01:14,250 --> 00:01:19,049

running on a web server and it exists

somewhere in the world and it's

16

00:01:19,049 --> 00:01:23,670

receiving connections and your browser

hero in this example is their client

17

00:01:23,670 --> 00:01:29,490

it's creating connections to the server

to receive content similarly if you can

18

00:01:29,490 --> 00:01:34,740

be used email before your email program

such as Outlook or Thunderbird

19

00:01:34,740 --> 00:01:40,740

that's a client it opens connections and

it gives information to the user whereas

20

00:01:40,740 --> 00:01:46,259

the server is generally run by some big

company for you and that exists

21

00:01:46,259 --> 00:01:50,850

somewhere on the internet in cyberspace

the important distinction is that the

22

00:01:50,850 --> 00:01:56,670

server is running for a long time

it generally somebody starts that but it

23

00:01:56,670 --> 00:02:01,740

might run for years at a time receiving

connections and doing work where is the

24

00:02:01,740 --> 00:02:05,610

client like your web browser you have it

open for maybe 10 15 minutes but then

25

00:02:05,610 --> 00:02:09,149

you close it again it's a very short

lived thing it's a very important

26

00:02:09,149 --> 00:02:12,060

distinction as we go forward because

it's always important to keep in mind

27

00:02:12,060 --> 00:02:17,040

what is your server and

your client it might be straightforward

28

00:02:17,040 --> 00:02:22,800

for websites an email but you can get

muddled as we go on just a bit more

29

00:02:22,800 --> 00:02:28,230

theory before we get into the meat of it

there's a concept of ports that I feel

30

00:02:28,230 --> 00:02:31,520

like I need to mention as well and

you're probably familiar with ports

31

00:02:31,520 --> 00:02:37,140

actually maybe maybe not

these are just numbers that we use to

32

00:02:37,140 --> 00:02:42,900

say what services are running where so

for instance number 80 port 80 is what

33

00:02:42,900 --> 00:02:49,620

we use for website connections or HTTP

connections we also have for instance

34

00:02:49,620 --> 00:02:55,290

port 25 or email there's a list of these

port numbers you can look at but the

35

00:02:55,290 --> 00:02:58,709

important thing to know is depending on

what service you want to connect to

36

00:02:58,709 --> 00:03:03,720

whether it's email or HTTP for websites

or something you're inventing yourself

37

00:03:03,720 --> 00:03:10,350

you need to have a port number so to

make an analogy do we look at these just

38

00:03:10,350 --> 00:03:14,850

like when back in the day with just two

basic you would open a comm port for

39

00:03:14,850 --> 00:03:21,239

example open these comport come one come

two come three somewhat and I have to

40

00:03:21,239 --> 00:03:24,660

admit I'm probably not old enough to

every remember opening comports in

41

00:03:24,660 --> 00:03:25,700

QBasic unfortunately...

42

00:03:25,700 --> 00:03:28,769

- Okay thank

you Luke for being here with us it was a

43

00:03:28,769 --> 00:03:31,740

pleasure... calling us old, keep on

44

00:03:31,740 --> 00:03:37,080

- Also, going to the main of a theme mechanics

45

00:03:37,080 --> 00:03:41,340

of how you work with these tcp/ip

connections are pretty similar to

46

00:03:41,340 --> 00:03:50,130

comports and we ever use some of the

same commands for doing it I think the

47

00:03:50,130 --> 00:03:54,450

analogy probably breaks down when we

start talking about port numbers and

48

00:03:54,450 --> 00:03:57,540

that's a good thing you brought it up

because it might confuse some people

49

00:03:57,540 --> 00:04:02,400

actually when we talk about come one

come two come three when generally

50

00:04:02,400 --> 00:04:06,239

talking about different devices see if

you talk on com1 you might be talking to

51

00:04:06,239 --> 00:04:10,769

your modem whereas com2 might be your

mouse or your I don't know what else to

52

00:04:10,769 --> 00:04:15,290

people connect to comport Sprinter maybe

I know when I go on printer ports me

53

00:04:15,290 --> 00:04:21,900

where when we deal with TCP connections

will generally find that the IP address

54

00:04:21,900 --> 00:04:27,420

of a host name is more talking about the

device whereas the port number

55

00:04:27,420 --> 00:04:31,500

tends to be more about what kind of

service you'd like to access so again

56

00:04:31,500 --> 00:04:37,560

we'll see this maybe a bit later just to

keep in mind that ports in TCP are not

57

00:04:37,560 --> 00:04:43,110

really the same thing as ports on a

comport sense okay but that's a good

58

00:04:43,110 --> 00:04:46,800

point thanks for bringing it up I

thought I'd go on too much with theory

59

00:04:46,800 --> 00:04:52,050

so I thought I just start with a really

simple example and I can work through

60

00:04:52,050 --> 00:05:01,590

how Warner and bolts are operating so to

load a web page I feel like this is

61

00:05:01,590 --> 00:05:04,880

something that people would probably

want to do it they're trying to maybe

62

00:05:04,880 --> 00:05:10,320

fetch data from a website maybe its

stock price information or trying to

63

00:05:10,320 --> 00:05:14,400

find out what the latest version of QB64 is it's a pretty important thing I

64

00:05:14,400 --> 00:05:18,900

think so

I this is podcasts of course I can't

65

00:05:18,900 --> 00:05:22,560

really show the coding in great detail

but all the chord samples you'll be

66

00:05:22,560 --> 00:05:25,830

mentioning will be in the description of

this episode oh I get that that's

67

00:05:25,830 --> 00:05:28,110

excellent

so if you're following along at home

68

00:05:28,110 --> 00:05:32,220

with the code samples there's a sample

one there are a few samples so number

69

00:05:32,220 --> 00:05:36,810

them just raise your reference and to

actually open the connection and get

70

00:05:36,810 --> 00:05:40,860

some data from website you only needs

three commands and you probably already

71

00:05:40,860 --> 00:05:46,970

know two of them so the three commands

we use are open client get and put so

72

00:05:46,970 --> 00:05:53,100

\_OPENCLIENT is the new special

underscore commands that's the QB64

73

00:05:53,100 --> 00:05:58,080

addition to the old QBasic command set

and although the syntax looks a little

74

00:05:58,080 --> 00:06:02,700

strange it never really changes that

much so you can always copy and paste it

75

00:06:02,700 --> 00:06:08,370

from the wiki the open client takes a

single string and it tells open row

76

00:06:08,370 --> 00:06:12,750

tells the computer really where to

connect to so when you're using our web

77

00:06:12,750 --> 00:06:19,530

browser you typing in address google.com

something like that you pass a very

78

00:06:19,530 --> 00:06:23,340

similar thing to \_OPENCLIENT except you

also need to add in the port number

79

00:06:23,340 --> 00:06:28,020

because of the web browser is always

displaying web pages it can just assume

80

00:06:28,020 --> 00:06:32,550

the port number QB64 can't assume the

port number because it doesn't know what

81

00:06:32,550 --> 00:06:36,360

you want to do it's more flexible you

can see in the open client and sample

82

00:06:36,360 --> 00:06:40,959

one there we're saying we'd like a

tcp/ip connection to port 80 or

83

00:06:40,959 --> 00:06:47,629

alephc.xyz which is just an example

domain that I had start for this talk

84

00:06:47,629 --> 00:06:51,619

today that is a real domain if you want

to go to its and try and load pages that

85

00:06:51,619 --> 00:06:56,990

will work but it just gives a really

simple response so anybody with running

86

00:06:56,990 --> 00:07:02,089

the code can see something happen easily

the next two commands that we need to

87

00:07:02,089 --> 00:07:08,089

get input they're very similar to how

they operate with comports I believe and

88

00:07:08,089 --> 00:07:13,039

definitely very similar to how I operate

with binary files so if you've used them

89

00:07:13,039 --> 00:07:18,259

for reading and writing general binary

data you're probably already got a

90

00:07:18,259 --> 00:07:24,139

feeling of how they work but if for

those who don't get input put will write

91

00:07:24,139 --> 00:07:30,469

a variable to a file handle where as

guest will read in dust or a file handle

92

00:07:30,469 --> 00:07:36,409

and store it in a variable for you oh

and just to backtrack a bit open client

93

00:07:36,409 --> 00:07:43,059

it returns a handle like file handle so

if you've done open file for binary as

94

00:07:43,059 --> 00:07:48,110

one open client is very similar gives

you back a number but you can store in a

95

00:07:48,110 --> 00:07:52,669

variable and pass around everywhere so

that's what we've called H at this

96

00:07:52,669 --> 00:07:58,849

example there's a bit of complexity here

for HTTP itself so if you've ever done

97

00:07:58,849 --> 00:08:02,959

any kind of work with HTTP you might be

familiar with its format but if you

98

00:08:02,959 --> 00:08:07,099

haven't don't worry it's pretty simple

all we need to do is send the server a

99

00:08:07,099 --> 00:08:11,209

string that says what file we'd like to

get back so in this case we're going to

100

00:08:11,209 --> 00:08:15,979

try and load test dot txt and if you're

looking at the sample there in detail

101

00:08:15,979 --> 00:08:21,169

you can see on the second line we just

say get test.txt and we're using HTTP

102

00:08:21,169 --> 00:08:26,329

1.0 it's the version identifier and you

have some carriage returns and line feeds

103

00:08:26,329 --> 00:08:31,639

to finish it off then we put it to the

server that string and we wait for our

104

00:08:31,639 --> 00:08:36,439

response and then just so they to prove

that we receive some data we can print

105

00:08:36,439 --> 00:08:39,079

that to the screen and that's really all

there is to it

106

00:08:39,079 --> 00:08:42,979

that's really the it's a simplest

program you can write with tcp/ip

107

00:08:42,979 --> 00:08:47,000

connections but that's pretty effective

if you're just trying to load a small

108

00:08:47,000 --> 00:08:52,089

amount of data from a website it work so

the good thing about these

109

00:08:52,089 --> 00:08:54,800

communications online is that even

though

110

00:08:54,800 --> 00:09:01,399

are going on over the Internet

the the way to get the data and to

111

00:09:01,399 --> 00:09:06,200

actually send it it's pretty simple and

it's actually very similar to what we do

112

00:09:06,200 --> 00:09:10,370

to a file on the disk, right? That's right

there are of course complications

113

00:09:10,370 --> 00:09:15,829

because networking is a more complicated

thing but it should be very familiar to

114

00:09:15,829 --> 00:09:21,140

people who have worked with binary data

files before moving on I've got sample

115

00:09:21,140 --> 00:09:25,190

two as well and sample two looks a

little longer but it's really just to

116

00:09:25,190 --> 00:09:29,420

illustrate an important point that a lot

of people miss when they're starting to

117

00:09:29,420 --> 00:09:34,040

do this networking stuff and it tends to

catch people out so I thought it worth

118

00:09:34,040 --> 00:09:40,940

mentioning TCP is what we call a stream

protocol and that basically means that

119

00:09:40,940 --> 00:09:46,579

it doesn't really know about the start

and end of data that you said it's

120

00:09:46,579 --> 00:09:51,560

mushes it all together it's using an

analogy in fact named stream is a pretty

121

00:09:51,560 --> 00:09:56,480

good sense of analogy if you tip a

bucket of water in a stream and then you

122

00:09:56,480 --> 00:10:00,140

add another bucket of water you can't

see where the first bucket starter than

123

00:10:00,140 --> 00:10:04,970

the second bucket started it all just

becomes one constant flow of water in

124

00:10:04,970 --> 00:10:11,029

the same way if we send a bit of data

via TCP and then we send a bit more data

125

00:10:11,029 --> 00:10:15,500

the other end doesn't know where the

boundary between those two is it just

126

00:10:15,500 --> 00:10:21,770

sees it all as a continuous stream of

characters there's no way to know when

127

00:10:21,770 --> 00:10:27,529

data stopped coming that's right um yeah

so we just see characters coming in and

128

00:10:27,529 --> 00:10:34,040

we can't tell what the number of puts

that's a person on the other end used

129

00:10:34,040 --> 00:10:37,820

and we don't know necessarily when

they're finished just by looking which

130

00:10:37,820 --> 00:10:42,649

means that we need some sort of strategy

for dealing with that which talked about

131

00:10:42,649 --> 00:10:48,440

maybe in a moment that just a so so a

sample too is demonstrating it's exactly

132

00:10:48,440 --> 00:10:54,110

the same a sample one except that

instead of sending the HTTP request in

133

00:10:54,110 --> 00:10:59,990

one push I'm now sending it with two

puts and the big takeaway is that if I

134

00:10:59,990 --> 00:11:03,260

hated yeah call the same because the

server and the other end doesn't know

135

00:11:03,260 --> 00:11:07,550

that we've used two puts it receives

some characters and then it receives

136

00:11:07,550 --> 00:11:12,050

some more character

and it simply thinks that there may be

137

00:11:12,050 --> 00:11:16,100

they've been delayed by networking

issues or just by a slow computer

138

00:11:16,100 --> 00:11:22,310

somewhere it can't tell that we've sent

it as two separate puts show you back to

139

00:11:22,310 --> 00:11:26,210

the binary file analogy it's probably

not too different to binary files

140

00:11:26,210 --> 00:11:30,470

actually and that if you put one thing

and then you put another thing it's just

141

00:11:30,470 --> 00:11:33,950

been they'd become right next to each

other and you can't tell where one was

142

00:11:33,950 --> 00:11:37,940

and where the other one was especially

if you're dealing with strings this is

143

00:11:37,940 --> 00:11:41,390

useful if you want to just send that a

bit by bit but it makes it hard if

144

00:11:41,390 --> 00:11:44,540

you're trying to receive data which

we'll get to bit later we talked about

145

00:11:44,540 --> 00:11:49,190

servers but as Felipa mentioned the

several so doesn't know when we finished

146

00:11:49,190 --> 00:11:54,350

ending data which is why for HTTP we

have to send carriage return line feed

147

00:11:54,350 --> 00:11:59,630

carriage return line feed

that's the HTTP approach different

148

00:11:59,630 --> 00:12:03,980

systems use different techniques to

signal when the data is finished so if

149

00:12:03,980 --> 00:12:09,920

using IRC for instance the IRC protocol

is in the chat protocol they just say as

150

00:12:09,920 --> 00:12:17,150

soon as you see a new line at the end of

the message if you're using say a more

151

00:12:17,150 --> 00:12:21,410

binary based format I'm using text

strings if you're doing pads in game

152

00:12:21,410 --> 00:12:27,380

server they might say well you always

send 23 bytes and so you'll send 23

153

00:12:27,380 --> 00:12:31,670

bytes and a server will count and once

it's received 23 bytes it will say yes

154

00:12:31,670 --> 00:12:35,420

I've received the whole message that

works too if they're not dealing with

155

00:12:35,420 --> 00:12:40,880

text-based data say if you're sending

maybe positions of players on a map or

156

00:12:40,880 --> 00:12:46,520

high scores that can be an effective way

of doing things - yeah I mean if you if

157

00:12:46,520 --> 00:12:50,480

you if it does get split up or there's a

delay or something happens can anything

158

00:12:50,480 --> 00:12:55,850

else enter that stream before all the

way to the server do we have to worry

159

00:12:55,850 --> 00:13:00,200

about that on the programmer side so for

the progress point of view no you don't

160

00:13:00,200 --> 00:13:05,150

have to worry about that one of the

really nice things about tcp is it makes

161

00:13:05,150 --> 00:13:08,840

some guarantees to the programmer that

it can rely on and so if guarantees that

162

00:13:08,840 --> 00:13:13,940

your data will arrive and just leave

someone cuts the internet cable at cards

163

00:13:13,940 --> 00:13:17,960

but it'll resend data automatically for

you if it doesn't arrive whatever the

164

00:13:17,960 --> 00:13:22,080

reason it makes it arise in order you

make sure there's no junk

165

00:13:22,080 --> 00:13:26,370

that gets in between from random network

sources as best as it can and it makes

166

00:13:26,370 --> 00:13:29,610

sure the data is error-free so it does

error correction for you automatically

167

00:13:29,610 --> 00:13:35,250

all handled by the underlying operating

system the programmer can just send data

168

00:13:35,250 --> 00:13:40,590

as they want and not have to worry about

any of that set okay maybe there are

169

00:13:40,590 --> 00:13:44,580

security concerns that will talk about

security I think at the end of this

170

00:13:44,580 --> 00:13:47,100

segment

good know for the most part it's all

171

00:13:47,100 --> 00:13:50,670

taken care of for you

let's talk about the other side of the

172

00:13:50,670 --> 00:13:55,920

equation now the server side what was

doing been doing at the moment we've

173

00:13:55,920 --> 00:13:58,560

been talking through servers that

already exist on the Internet

174

00:13:58,560 --> 00:14:03,120

and they're generally written by people

who spend a lot of time and effort to

175

00:14:03,120 --> 00:14:08,550

produce really complicated servers but

we can do our own in QB 64 with just two

176

00:14:08,550 --> 00:14:12,360

new commands so the two new commands

where you using our open host and open

177

00:14:12,360 --> 00:14:18,120

connections it can be a bit tricky for

people in being introduced to this to

178

00:14:18,120 --> 00:14:21,840

understand those differences between

open hosts an open connection so I'll

179

00:14:21,840 --> 00:14:27,210

jump straight into the example and

explain the based on that so we begin

180

00:14:27,210 --> 00:14:32,700

with the open host command always and

the format of it looks pretty simple but

181

00:14:32,700 --> 00:14:37,140

pretty similar to the open client that

we saw earlier the only thing you'll

182

00:14:37,140 --> 00:14:42,360

notice is that we no longer give it a

website address so no IP address or

183

00:14:42,360 --> 00:14:46,440

domain name

we're just saying open host tcp/ip and

184

00:14:46,440 --> 00:14:52,020

then a port number here I've chosen to

use port 9991 you can

185

00:14:52,020 --> 00:14:56,700

really pick any port you want on your

computer for your service you generally

186

00:14:56,700 --> 00:14:59,820

have to avoid the ones below about a

thousand twenty-four because of the

187

00:14:59,820 --> 00:15:04,790

permissions issues but aside from that

you can really pick anything up to about

188

00:15:04,790 --> 00:15:11,490

whatever the limit of an integer data

type is these five thousand I think big

189

00:15:11,490 --> 00:15:16,320

birthday probably knows better than I do

yeah I don't know exactly the limit but

190

00:15:16,320 --> 00:15:21,420

I've used pretty high numbers just to

avoid collision with other valid things

191

00:15:21,420 --> 00:15:26,400

and if you look online you can find some

common ports and it's not that hard to

192

00:15:26,400 --> 00:15:31,560

find so you can just avoid those and go

with a random number if you choose I

193

00:15:31,560 --> 00:15:35,850

guess that's right so I've chosen nine

nine nine one

194

00:15:35,850 --> 00:15:39,089

I'm pretty sure that nothing in my

computer be using it as long as you take

195

00:15:39,089 --> 00:15:42,750

a nice high numbered port there's very

good chance that it'll be perfectly fine

196

00:15:42,750 --> 00:15:45,620

that's it

doing an open host doesn't actually

197

00:15:45,620 --> 00:15:53,339

start talking to any remote computers it

really just says to QB64 I'd like to

198

00:15:53,339 --> 00:15:59,490

listen for people trying to connect to

port 9 9 91 so when the client does open

199

00:15:59,490 --> 00:16:05,009

client 1 point 9 9 9 1 that's what we're

listening out for but we haven't

200

00:16:05,009 --> 00:16:08,399

actually started talking to anyone yet

but it gives us back what we call the

201

00:16:08,399 --> 00:16:12,870

host handle and that the only thing we

can do with that host handle is then

202

00:16:12,870 --> 00:16:18,690

pass it to open connection so when

somebody tries to connect to us us being

203

00:16:18,690 --> 00:16:22,319

the server when a client tries to

connect to us we don't start talking to

204

00:16:22,319 --> 00:16:25,860

them straight away they go into a queue

like if you go to the bank and you have

205

00:16:25,860 --> 00:16:30,750

to wait in queue to talk to person at

the front desk and by calling open

206

00:16:30,750 --> 00:16:34,889

connection we're saying we'd like to

speak to the next person in the queue so

207

00:16:34,889 --> 00:16:39,899

once we call open connection it gives us

back a number and if the number is not

208

00:16:39,899 --> 00:16:43,949

equal to zero it means that yes there

was somebody in the queue and we're now

209

00:16:43,949 --> 00:16:47,670

talking to them if it had returned zero

that's it's a way of saying actually

210

00:16:47,670 --> 00:16:50,579

there's no one in the queue at all try

again later

211

00:16:50,579 --> 00:16:54,750

but we assume that we have someone in

the key who is waiting to talk to us and

212

00:16:54,750 --> 00:16:59,399

now we've got back our connection hand

or fronting connection and we can talk

213

00:16:59,399 --> 00:17:03,269

to them using the same get input

commands that we were using earlier so

214

00:17:03,269 --> 00:17:07,620

here we get a value from the client we

print out that value to our local screen

215

00:17:07,620 --> 00:17:11,520

and then we double it and send the

result back to the client

216

00:17:11,520 --> 00:17:16,409

so all this server does is receive

numbers and send back what twice that

217

00:17:16,409 --> 00:17:22,049

number is it's a pretty useful service

all things considered but real services

218

00:17:22,049 --> 00:17:26,250

are much more complicated than this they

just do more stuff with the data before

219

00:17:26,250 --> 00:17:30,659

sending it back so this makes a pretty

good example of all the basic techniques

220

00:17:30,659 --> 00:17:34,919

we're going to use one thing you might

notice is we've got a do loop around the

221

00:17:34,919 --> 00:17:40,440

gate come up and this is where we really

differ from dealing with binary files

222

00:17:40,440 --> 00:17:46,020

when you're working with a binary file

the data is always there it's on disk

223

00:17:46,020 --> 00:17:49,590

it's stored the computer will fetch it

when you ask for it we

224

00:17:49,590 --> 00:17:56,220

tcp/ip networking sometimes we have to

wait for the data to arrive it may be

225

00:17:56,220 --> 00:18:00,540

because the network is slow or because

the client is taking a little while to

226

00:18:00,540 --> 00:18:04,800

actually send the data in the first

place well whatever the reason is we

227

00:18:04,800 --> 00:18:08,610

have to actually wait for some dad to

arrive before we can work with it so if

228

00:18:08,610 --> 00:18:14,910

we loop over the get command and we

check that the AOS condition which it

229

00:18:14,910 --> 00:18:19,470

might be a little bit confusing because

COF is generally meaning the end of data

230

00:18:19,470 --> 00:18:24,960

when you're dealing with a binary file

it's been refilled a little bit for TCP

231

00:18:24,960 --> 00:18:28,680

connections it's a little backwards

right yeah

232

00:18:28,680 --> 00:18:33,690

I don't I try to think about too hard I

just know that I have to live while iOS

233

00:18:33,690 --> 00:18:40,550

of the connection and that just loops

until we get actually readings and data

234

00:18:40,550 --> 00:18:46,380

it's almost this something I copy-paste

at this point and I would recommend that

235

00:18:46,380 --> 00:18:50,910

if you're dealing with this you'd just

copy that line as well you can look at

236

00:18:50,910 --> 00:18:56,460

the exact semantics of the EOS for TCP

connections that it's almost not worth

237

00:18:56,460 --> 00:19:01,530

going into too much we'll just say that

that loop loops over the guest until

238

00:19:01,530 --> 00:19:05,700

they get receives data and once the

texter receive data the loop exits so

239

00:19:05,700 --> 00:19:08,880

that way when we get to the print line

we can be sure we're actually printing a

240

00:19:08,880 --> 00:19:12,990

data we received from the client we can

double it and then we can just use put

241

00:19:12,990 --> 00:19:17,610

to send it back to the client again

pretty similar to what we're doing when

242

00:19:17,610 --> 00:19:23,490

we're writing the client itself so

you're sending a value here this is a

243

00:19:23,490 --> 00:19:29,250

long variable so this is four bytes and

this is low ASCII so there's no problem

244

00:19:29,250 --> 00:19:34,530

sending those over TCP I imagine because

you can download files right so no

245

00:19:34,530 --> 00:19:38,160

problem no limitations on the type of

data you're sending it doesn't have to

246

00:19:38,160 --> 00:19:43,260

be a string right that's right that's a

good question to bring up you can send

247

00:19:43,260 --> 00:19:48,990

most kinds of data over TCP connections

so any of the numeric data types really

248

00:19:48,990 --> 00:19:52,710

easy to send because they have a fixed

number of bytes so here we're sending

249

00:19:52,710 --> 00:19:57,900

along that's always four bytes and you

might recall earlier on I talked about

250

00:19:57,900 --> 00:20:01,530

different strategies for knowing and

received all the data

251

00:20:01,530 --> 00:20:06,300

get knows that value is 4 bytes so it's

going to read 4 bytes and as long as we

252

00:20:06,300 --> 00:20:10,560

all agree on using a long data type we

can all agree that we have to read and

253

00:20:10,560 --> 00:20:17,070

write 4 bytes and everything's fine all

numeric values are fine strings as we

254

00:20:17,070 --> 00:20:21,570

saw earlier we can work with we can even

send an egg send and get entire arrays

255

00:20:21,570 --> 00:20:28,500

at once if we wanted to so if you had an

array of 100 long integers we could put

256

00:20:28,500 --> 00:20:33,060

the entire array I don't have any

examples of that but maybe that's useful

257

00:20:33,060 --> 00:20:36,870

if you're doing some sort of game style

programming I've got the only things you

258

00:20:36,870 --> 00:20:43,380

can't put and get over a TCP connection

is min blocks not natively anyway you'd

259

00:20:43,380 --> 00:20:47,040

have to write your own loop if you want

to do with an M block and also a raise

260

00:20:47,040 --> 00:20:53,220

of strings cause problems because it

doesn't know how long each string is so

261

00:20:53,220 --> 00:20:57,810

you have to avoid those but most other

data types it'll just work because it

262

00:20:57,810 --> 00:21:02,130

can know how big they are oh and a

user-defined types as well no works is

263

00:21:02,130 --> 00:21:04,350

fine

so once we use temp the data to the

264

00:21:04,350 --> 00:21:10,260

client we can just close a connection

that's the clothes see again very much

265

00:21:10,260 --> 00:21:13,020

like you could billing with a file once

you're finished reading or writing from

266

00:21:13,020 --> 00:21:16,800

the file it's important to a closer

connection or close the file so that we

267

00:21:16,800 --> 00:21:21,390

can free any resources that are

associated with it in the operating and

268

00:21:21,390 --> 00:21:25,470

all we do as a server is loop

indefinitely so we have an infinite loop

269

00:21:25,470 --> 00:21:29,190

that just constantly waits for new

connections or checks for new

270

00:21:29,190 --> 00:21:33,420

connections in the queue processes of

closes them and then moves on to the

271

00:21:33,420 --> 00:21:37,800

next one really all there is to writing

a server we can get more complicated of

272

00:21:37,800 --> 00:21:42,480

course but this really illustrates all

the basic ideas I've also gotten sample

273

00:21:42,480 --> 00:21:47,220

for an example of a client that could

use this long doubler there's nothing

274

00:21:47,220 --> 00:21:50,840

really too interesting in here that we

haven't seen in the previous examples

275

00:21:50,840 --> 00:21:55,800

you can see we're using open clients

actually maybe the localhost might be

276

00:21:55,800 --> 00:21:59,550

worth talking about in the earlier

examples we were talking to a web

277

00:21:59,550 --> 00:22:03,960

address like what you've typed into your

website into your web browser ok if we

278

00:22:03,960 --> 00:22:07,620

want to just talk to programs on the

same computer we can use a special

279

00:22:07,620 --> 00:22:12,330

address called localhost and Nick the

operating system understands localhost

280

00:22:12,330 --> 00:22:16,560

always just mean talk to

yourself the name suggests it's the

281

00:22:16,560 --> 00:22:21,570

local machine which is really useful if

I'm talk to talk between two programs on

282

00:22:21,570 --> 00:22:25,380

the same computer like we're doing this

demo so you don't actually need two

283

00:22:25,380 --> 00:22:28,920

computers to run this client and server

you can run them both on your desktop

284

00:22:28,920 --> 00:22:36,210

other than that we put our value as a

long to the handle into the hand we get

285

00:22:36,210 --> 00:22:40,110

back to our open client and then we have

that same loop with the guest to receive

286

00:22:40,110 --> 00:22:44,070

back the data we don't know how long

it'll take for the server process our

287

00:22:44,070 --> 00:22:49,200

data so we have to do this loop while

AOS to wait for the data to be returned

288

00:22:49,200 --> 00:22:55,230

and that's really all there is to it yes

just looking at this example with both

289

00:22:55,230 --> 00:23:02,480

the server here and the client code it's

really weird to look at these gasline

290

00:23:02,480 --> 00:23:09,270

trying to read into a long variable in a

loop this is really weird so we don't

291

00:23:09,270 --> 00:23:14,460

have to worry here if both parties have

agreed on the type of response I'm

292

00:23:14,460 --> 00:23:19,380

really going to only read here into this

variable Cuba 64 manages that for me

293

00:23:19,380 --> 00:23:22,620

right I don't have to worry about that

that's right so although it's in a loop

294

00:23:22,620 --> 00:23:29,130

and it does look at these reads the

combination of the wild AR where means

295

00:23:29,130 --> 00:23:35,220

that all we're really doing is one read

that will succeed so the loop is to

296

00:23:35,220 --> 00:23:39,290

handle all the reads that don't 16

because the data hasn't arrived yet

297

00:23:39,290 --> 00:23:45,600

this combination of commands really

guarantees that we read exactly one long

298

00:23:45,600 --> 00:23:50,190

from the network and as long as we've

agreed with the other person we're

299

00:23:50,190 --> 00:23:54,510

talking to that they don't just had one

long that will work properly for us so

300

00:23:54,510 --> 00:24:00,180

that after the EOS lines if you had a

longer data and you hadn't agreed like

301

00:24:00,180 --> 00:24:05,070

terminator you'd want to check for that

too before you actually close and do

302

00:24:05,070 --> 00:24:10,830

everything yes so if you are writing a

web server that uses HTTP like we were

303

00:24:10,830 --> 00:24:14,010

talking to you know earlier examples you

have to do something a little bit

304

00:24:14,010 --> 00:24:17,730

different you'd have to read in the data

and probably append it to a string and

305

00:24:17,730 --> 00:24:22,080

at the same time you'd be checking to

see if that string has the carriage

306

00:24:22,080 --> 00:24:26,700

return line feed carriage return line

feed in it okay I've done some tcp/ip

307

00:24:26,700 --> 00:24:31,050

communications

locally before and I've always come up

308

00:24:31,050 --> 00:24:38,070

with a costume for months I mean I

always have an and marker and I agree

309

00:24:38,070 --> 00:24:43,980

with myself on what to sand and then I

send some information beforehand saying

310

00:24:43,980 --> 00:24:50,070

hey I'm sending you these many bytes of

data and then the the client and the

311

00:24:50,070 --> 00:24:53,940

server agree on that and then there is

an end marker so in this approach I

312

00:24:53,940 --> 00:25:00,600

usually do a loop until there is data

coming and then I keep adding the the

313

00:25:00,600 --> 00:25:06,780

data that's coming into a longer array

which then I parse is that maybe too

314

00:25:06,780 --> 00:25:11,220

much of her hand you think no we can

come up with more complex schemes and

315

00:25:11,220 --> 00:25:16,230

often or sometimes once they often we do

need the more complicated schemes to

316

00:25:16,230 --> 00:25:23,390

transfer more complicated kinds of data

so you're using a Heather style

317

00:25:23,390 --> 00:25:27,000

technique where we tell the client

hammer Siddartha we're going to send

318

00:25:27,000 --> 00:25:31,860

them and then we send them that data and

that can be very useful if we need to

319

00:25:31,860 --> 00:25:35,940

send the variable amounts they in our

simple examples here we're always

320

00:25:35,940 --> 00:25:41,490

sending the same four bytes so it was

easy but if you're dealing with things

321

00:25:41,490 --> 00:25:45,000

where you don't know ahead of time and

how much data you have to send that's a

322

00:25:45,000 --> 00:25:49,890

very decent way to do it by sending a

kind of header and also the end of

323

00:25:49,890 --> 00:25:55,830

message marker is a useful technique

sometimes the end of message market

324

00:25:55,830 --> 00:26:00,030

technique doesn't always work because it

can be hard to know what's the end of

325

00:26:00,030 --> 00:26:05,370

message marker versus what actual

message data so depending on what your

326

00:26:05,370 --> 00:26:10,350

message is that may or may not be a

viable method for instance field

327

00:26:10,350 --> 00:26:14,700

downloading a file you don't want your

file to contain your end of message

328

00:26:14,700 --> 00:26:19,080

marker because then the receiving end

would think the message has ended but

329

00:26:19,080 --> 00:26:23,610

actually there's more file content to

come so when we're downloading files

330

00:26:23,610 --> 00:26:29,910

with say HTTP it does the header file

where it tells us how much data is going

331

00:26:29,910 --> 00:26:33,900

to come it says there will be five

thousand bytes and then since five

332

00:26:33,900 --> 00:26:39,390

thousand bytes so they're all they're

all valid approaches to take and we can

333

00:26:39,390 --> 00:26:43,450

use one or a combination of them

depends on what the kind of data we're

334

00:26:43,450 --> 00:26:49,809

sending is a few other miscellaneous

points I think are worth mentioning

335

00:26:49,809 --> 00:26:57,070

after once I handle multiple clients at

the same time it's a little bit harder

336

00:26:57,070 --> 00:27:01,750

just in terms of the programming

footwork that the TCP concepts don't

337

00:27:01,750 --> 00:27:06,669

become much more complicated we always

have one host handle that we get back

338

00:27:06,669 --> 00:27:12,429

from open hosts but we can have many

connection handles we get back on open

339

00:27:12,429 --> 00:27:16,179

connection because each time we call

open connection we're potentially

340

00:27:16,179 --> 00:27:22,390

opening a connection to a new client so

if we want to have very long running

341

00:27:22,390 --> 00:27:26,740

connections maybe we're running a game

service or some sort of multiplayer game

342

00:27:26,740 --> 00:27:32,169

with what people will be able to enjoin

and leave the game as they wish we can

343

00:27:32,169 --> 00:27:36,789

call open connection regularly as we're

doing game processing and see is there

344

00:27:36,789 --> 00:27:40,630

anybody who wants to join the game and

that will give us back a different

345

00:27:40,630 --> 00:27:45,100

connection handle and then we just have

to loop over all our connection handles

346

00:27:45,100 --> 00:27:49,510

and do a get to see if there's data

that's arrived from any of our players

347

00:27:49,510 --> 00:27:54,549

or any of our clients yes in general we

can use this for so I mentioned the

348

00:27:54,549 --> 00:27:58,779

start we've seen we can download file

from internet which is pretty useful by

349

00:27:58,779 --> 00:28:04,870

itself I think if we want to do email an

email isn't much more complicated it

350

00:28:04,870 --> 00:28:07,890

just means that there's a few more

commands we have to send back and forth

351

00:28:07,890 --> 00:28:13,299

but it's really just combinations of

doing put and in a string and then doing

352

00:28:13,299 --> 00:28:16,450

a get to receive the message from the

server yeah the thing about these

353

00:28:16,450 --> 00:28:21,940

different approaches is that on the cuba

64 side it's only open client open

354

00:28:21,940 --> 00:28:25,929

connection open host get and put what

you have to deal with her the different

355

00:28:25,929 --> 00:28:29,500

protocols for the whatever type of

communication we really want to do rain

356

00:28:29,500 --> 00:28:35,200

that's right so TCP it's what we call

layer 4 and I don't complicate things

357

00:28:35,200 --> 00:28:40,690

work for means but tcp/ip works at one

layer and all these protocols that we

358

00:28:40,690 --> 00:28:47,380

know and love with the web stuff email

chat they offer at one level higher so

359

00:28:47,380 --> 00:28:51,280

our QB 64 program has to know the

details of that protocol that it wants

360

00:28:51,280 --> 00:28:55,000

to talk with and that's on the program

is implement that

361

00:28:55,000 --> 00:29:00,760

fortunately we don't have any built-in

commands that can speak HTTP or email

362

00:29:00,760 --> 00:29:06,040

natively but of course there's also

things like chat applications or network

363

00:29:06,040 --> 00:29:11,620

games kobold I know you're pretty big on

the games department but have you ever

364

00:29:11,620 --> 00:29:16,300

considered writing and network games

that could comply over into it I've got

365

00:29:16,300 --> 00:29:21,430

my battleship up on the forums but me

and Steve were trying to get it to work

366

00:29:21,430 --> 00:29:24,790

through his server he had set up and we

just couldn't get the thing to connect

367

00:29:24,790 --> 00:29:28,120

to itself never could figure it out that

might be something for us to look at

368

00:29:28,120 --> 00:29:32,200

maybe I reckon we could do some luck

there because it works on the local

369

00:29:32,200 --> 00:29:36,460

network just fine

ah that sounds like it might be some

370

00:29:36,460 --> 00:29:41,020

sort of firewall issue I don't want to

make this talk of the two longer there

371

00:29:41,020 --> 00:29:44,770

are always complications and it comes to

firewall how they can get in the way of

372

00:29:44,770 --> 00:29:48,610

people trying to play over the Internet

yeah we tried a few different things but

373

00:29:48,610 --> 00:29:54,430

yeah it might be that or maybe I didn't

have the right port accessing for some

374

00:29:54,430 --> 00:29:59,140

reason but you wouldn't mind getting it

working over the actual Internet that's

375

00:29:59,140 --> 00:30:03,040

kind of I put the networking stuff in

there but we have maybe a podcast

376

00:30:03,040 --> 00:30:08,110

episode where we all play battleship

together so one thing that maybe not

377

00:30:08,110 --> 00:30:11,290

particularly obvious what we can do a

system is really cool he is

378

00:30:11,290 --> 00:30:15,790

inter-process communication on the same

computer so if as we said if we have two

379

00:30:15,790 --> 00:30:19,930

programs running they can talk to each

other we simply say one's a client I'm

380

00:30:19,930 --> 00:30:24,040

on the server and so now instead of

having to muck around with writing

381

00:30:24,040 --> 00:30:27,670

temporary files to disk and having me I

have a program and reading temporary

382

00:30:27,670 --> 00:30:32,470

files and all of mess that creates we

can simply talk over the local network

383

00:30:32,470 --> 00:30:37,450

Philippe I think maybe you do this with

the B watch debugger you write actually

384

00:30:37,450 --> 00:30:43,420

I'm I intend to do that we'll be watch

with version 2.0 but I actually do that

385

00:30:43,420 --> 00:30:48,820

already with inform so when you're using

the editor for creating a new form with

386

00:30:48,820 --> 00:30:53,680

inform the the main module is

communicating with the form being edited

387

00:30:53,680 --> 00:30:59,170

via tcp/ip locally so I got that going

right that's really cool and that works

388

00:30:59,170 --> 00:31:04,090

well for you well pretty much what

actually happened before was that inform

389

00:31:04,090 --> 00:31:10,070

would

use files on the disk and then there was

390

00:31:10,070 --> 00:31:15,560

some overhead and writing to the disk

constantly was a concern and I even had

391

00:31:15,560 --> 00:31:20,240

some problems in Linux systems because

of the flushing to the disk

392

00:31:20,240 --> 00:31:25,490

but then tcp/ip solved all of that at

once one final thing to touch on I think

393

00:31:25,490 --> 00:31:28,910

and now talk about networking we would

be complete without talking about

394

00:31:28,910 --> 00:31:34,580

security everything we've done so far is

pretty basic and it works pretty well

395

00:31:34,580 --> 00:31:38,840

but it's important to remember that all

the connections we're running here

396

00:31:38,840 --> 00:31:43,880

they're not encrypted it anyway so all

the data we're sending and receiving can

397

00:31:43,880 --> 00:31:49,250

potentially be listened to by somebody

on the internet or even somebody just on

398

00:31:49,250 --> 00:31:54,230

the local land they were so inclined

which means that it's important to not

399

00:31:54,230 --> 00:31:59,810

send any really private data over this

kind of connection so you know certainly

400

00:31:59,810 --> 00:32:06,530

know things and details rather data that

kind of thing we don't have anything

401

00:32:06,530 --> 00:32:11,120

like HTTPS in the web browser where it

uses SSL encryption

402

00:32:11,120 --> 00:32:15,590

unfortunately you be 64 doesn't have

even the breeze in it at the moment do

403

00:32:15,590 --> 00:32:18,560

that kind of stuff although don't tell

anyone

404

00:32:18,560 --> 00:32:24,460

Lee birthdays just between me and you

guys hmm QB 64 maybe getting built-in

405

00:32:24,460 --> 00:32:29,390

SSL connection sometime in the new

future mm-hmm that's exciting is this

406

00:32:29,390 --> 00:32:32,780

off the record well it's off the record

but actually on the record if that makes

407

00:32:32,780 --> 00:32:36,770

sense so keep it in there okay I can

keep it I can't keep it okay that's good

408

00:32:36,770 --> 00:32:42,530

and then exciting more or less wraps up

all the networking stuff for one to show

409

00:32:42,530 --> 00:32:48,530

us today there's more complicated things

we can do but it's really just using the

410

00:32:48,530 --> 00:32:52,730

same tools we've already seen open

client open host open connection and

411

00:32:52,730 --> 00:32:57,680

then get input to send data back and

forth there are some good examples on

412

00:32:57,680 --> 00:33:04,190

the wiki of the leaves of a messenger

application check demo between two

413

00:33:04,190 --> 00:33:08,150

programs and there's also a demo one

there maybe we can make this in

414

00:33:08,150 --> 00:33:14,570

description of a more full-featured

subroutine for downloading files from

415

00:33:14,570 --> 00:33:18,570

websites you might like to check out and

use those

416

00:33:18,570 --> 00:33:23,340

I used that example from the wiki to

download files a lot I adapted here and

417

00:33:23,340 --> 00:33:28,920

there because it's pretty simple and

I've used it for example when four

418

00:33:28,920 --> 00:33:33,810

informs installer and updater I've used

the same routine that's available on the

419

00:33:33,810 --> 00:33:39,480

wiki with just a few minor adaptations

so it's pretty useful indeed Luke thank

420

00:33:39,480 --> 00:33:44,430

you so much for being here with us we

are doing this from three different time

421

00:33:44,430 --> 00:33:49,320

zones right now so it's a pleasure to be

able to use the internet for this type

422

00:33:49,320 --> 00:33:54,450

of interaction thank you so much for

shedding some light on this conversation

423

00:33:54,450 --> 00:34:00,240

about tcp/ip that it's happening over

tcp/ip thank you so much I hate the best

424

00:34:00,240 --> 00:34:04,770

the bubble but um this audio stuff is

probably using you they paid not to pay

425

00:34:04,770 --> 00:34:13,530

our shut up anyway anyway it's still

called voice what what does it call void

426

00:34:13,530 --> 00:34:18,720

be right boys ever cry pay yeah so if

you did pay over I paid I still got that

427

00:34:18,720 --> 00:34:26,610

going okay gentlemen it was a pleasure

all right been a pleasure being a good

428

00:34:26,610 --> 00:34:29,460

fun to be on the show thank you all for

listening

429

00:34:29,460 --> 00:34:36,169

this was QB64 report episode four and

I'll catch you all next time bye bye

430

00:34:47,929 --> 00:34:49,990

you