Subject: Socket Communication 101
Posted by nejnadusho on Fri, 07 Dec 2012 17:02:46 GMT
View Forum Message <> Reply to Message

Hi,

I need to figure out sockets.
And I have no much knowledge in that.

I run the reference examples and the communication works great and I understood what is going on both sides.

However, I still cannot imagine/assimilate how to transfer large chunks of data.

For example:
If a query to a database is returning an array of data how do I send it to the client side as one chunk?
Also how can I lable/mark that chunk to which class in my application is the receiver?

I already and I assume, but not sure, I can send an array with RawSend/RawRecv, then shoud I initialize/end RawOpen/RawClose the connection in a specific way?

Thank you very much.

Best,
Georgi

P.S. Please let me know if I should try to clarify myself.

Subject: Re: Socket Communication 101
Posted by nlneilson on Fri, 07 Dec 2012 17:20:31 GMT
View Forum Message <> Reply to Message

You question was clear.
What ever is sent over a socket, internet, is raw data AFAIK.
Just bits.
nejnadusho wrote on Fri, 07 December 2012 09:02However, I still cannot imagine/assimilate how to transfer large chunks of data.
Remove any \0 from the string you will be sending except at the end.

http://www.ultimatepp.org/forum/index.php?t=msg&th=7136& amp; amp;start=0&

Subject: Re: Socket Communication 101

Posted by nejnadusho on Fri, 07 Dec 2012 17:52:48 GMT
View Forum Message <> Reply to Message

Neil,

I am still very far from understanding how to send an array of information.

I mean I literally want to send an array through.

Something like

```
Array arr;

loop start{
  arr = fill with data
}

TcpSocket sock;

sock.RawSend(arr);
```

Is it possible or I am just a dreamer ?

Thank you very much.

Best,
Georgi

Subject: Re: Socket Communication 101
Posted by lectus on Fri, 07 Dec 2012 18:29:53 GMT
View Forum Message <> Reply to Message

I'm no expert in U++ or Sockets or even C++.

But this is what I think (correct me if I'm wrong):

Fill the array:

```
Array<String> FruitList;

FruitList.Add("Apple");
FruitList.Add("Banana");
FruitList.Add("Orange");
```

Design your own protocol and send data:

```
TcpSocket sock;
// Do some initialization of sock here

sock.Put("-- BEGIN FRUIT LIST --\n");

for(int i = 0; i < FruitList.GetCount(); i++)
sock.Put(FruitList[i] + '\n');

sock.Put("-- END FRUIT LIST --\n");
```

On the other side you have to treat what's received like:

```
String line = sock.GetLine();

if(line == "-- BEGIN FRUIT LIST --")
{
while(line != "-- END FRUIT LIST --")
{
line = sock.GetLine();
FruitList.Add(line);
}
}
```

I think to send a full array U++ would require some kind of serialization of objects. I don't know if it already has it.

Wait for someone more knowledgeable.

Subject: Re: Socket Communication 101
Posted by nlneilson on Fri, 07 Dec 2012 18:45:10 GMT
View Forum Message <> Reply to Message

```
sock.Put("-- BEGIN FRUIT LIST --\n");

for(int i = 0; i < FruitList.GetCount(); i++)
sock.Put(FruitList[i] + '\n');

sock.Put("-- END FRUIT LIST --\n");
```

Each sock.Put would be a different connect/disconnect, not good.

Add all the char to a single string or buf with the \0 at the end and send that.

One connect/disconnect

Subject: Re: Socket Communication 101
Posted by mirek on Fri, 07 Dec 2012 19:14:36 GMT
View Forum Message <> Reply to Message

nlneilson wrote on Fri, 07 December 2012 13:45sock.Put("-- BEGIN FRUIT LIST --\n");

for(int i = 0; i < FruitList.GetCount(); i++)
sock.Put(FruitList[i] + '\n');

sock.Put("-- END FRUIT LIST --\n");

Each sock.Put would be a different connect/disconnect, not good.

What makes you think that?

Mirek

Subject: Re: Socket Communication 101
Posted by lectus on Fri, 07 Dec 2012 19:41:59 GMT
View Forum Message <> Reply to Message

Since it's using TCP protocol, I believe there's a connection alive.

If you used UDP then yes, it doesn't use connections.

I think what you mean is that you can send it all in ONE chunk. Yes, it's also possible.

Maybe something like this:

```
String acc;
for(int i=0; i < FruitList.GetCount(); i++)
{
acc += FruitList[i];
acc += "-";
}
acc += '\n';

sock.Put(acc);
```

Then on the other side you can do:

```
String line = sock.GetLine();

String fruit;
for(int i=0;i < line.GetLength(); i++)
{
if (line[i] != '-')

else
{
FruitList.Add(fruit);
fruit="";
}
}
```

Data gets sent like this:
Quote:Apple-Banana-Orange

Subject: Re: Socket Communication 101
Posted by nlneilson on Fri, 07 Dec 2012 19:53:35 GMT
View Forum Message <> Reply to Message

In the for() loop
sock.Put(FruitList[i] + '\n');

it looks like a sock.Put is called each time through the loop or is each char sent through the same connection and then '\n' is sent after all the char been sent?

or is it ch \n ch \n ch \n

Also does String GetLine() stop at each \n or is it looking for \0

Maybe String GetAll(int len) will read a bunch of data including several \n until the max len is reached or \0

My client is in U++ and it will send a string with several \n and it is parsed after the server in the Java app.

edit: The last post by lectus where the data is added and that is sent is the way I do it.

Subject: Re: Socket Communication 101
Posted by lectus on Fri, 07 Dec 2012 19:57:10 GMT
View Forum Message <> Reply to Message

nlneilson wrote on Fri, 07 December 2012 14:53In the for() loop
sock.Put(FruitList[i] + '\n');

it looks like a sock.Put is called each time through the loop or is each char sent through the same connection and then '\n' is sent after all the char been sent?

or is it ch \n ch \n ch \n

Also does String GetLine() stop at each \n or is it looking for \0

Maybe String GetAll(int len) will read a bunch of data including several \n until the max len is reached or \0

My client is in U++ and it will send a string with several \n and it is parsed by the server in the Java app.

Each sock.Put sends a fruit name indexed as FruitList[i] with an ending \n.

It uses the same connection started by sock.Connect().

On the other side the sock.GetLine() reads everything until it finds \n and stores in a string without the \n.

Unless you're dealing with binary data I see no reason to not use strings ended with \n.

Subject: Re: Socket Communication 101
Posted by nlneilson on Fri, 07 Dec 2012 21:08:12 GMT
View Forum Message <> Reply to Message

lectus wrote on Fri, 07 December 2012 11:571. It uses the same connection started by sock.Connect().

2. On the other side the sock.GetLine() reads everything until it finds \n and stores in a string without the \n.

3. Unless you're dealing with binary data I see no reason to not use strings ended with \n.

1. That makes sense.

2. And that must be under while data waiting or similar.

3. I use \n but it is with the \0 that ends the send and receive.

The data is passed to the client to send as a string with \0
The server receives as a string with the \0

Nothing is added or combined in the client and nothing is parsed in the server. That is done with

error handling outside.

Less time in the socket and better error handling.

Subject: Re: Socket Communication 101
Posted by lectus on Fri, 07 Dec 2012 23:42:27 GMT
View Forum Message <> Reply to Message

Lower level sockets work with \0 and buffer sizes.

GetLine is a higher level function which parses the string until it finds \n. It's good for protocols based on lines of string such as HTTP protocol where you need to send a \n at the end.

Subject: Re: Socket Communication 101
Posted by nlneilson on Sat, 08 Dec 2012 01:36:30 GMT
View Forum Message <> Reply to Message

It has been several years since tinkering with different code for sockets so my memory is a bit fuzzy.

What you mentioned about "Lower level sockets work with \0 and buffer sizes" may be a clue as why \0 seems important to me.

I still do much of my parsing a character at a time rather than split at ','. The time difference now with faster CPUs is negligible.

I think it was back using Python for a client the Sleep time before data was out of order or garbled was 2 milli second on a dual core and 5 on a slower single core when using \0 as end of transmission. Any other way the time would be more. The server is on a thread and can accept several clients and have tried up to 5 without a problem except when trying to open and close the connections too fast. The amount of data sent doesn't make much difference as it is so fast from a few bytes to a few thousand as long as any parsing is not done in the server code.

There may be better ways like shared memory but that is above me.

It's good to go over stuff again and have my memory corrected when necessary.

Subject: Re: Socket Communication 101
Posted by lectus on Sat, 08 Dec 2012 14:37:16 GMT
View Forum Message <> Reply to Message

Yes, there's some optimization at CPU level when you compare a char with \0.

In ASM code a comparison happens like this:

```
CMP AX, ','
JZ FOUND_COMMA
.
.
.
FOUND_COMMA:
; Do something...
```

When you compare to zero it happens like this:

```
CMP AX, 0
JZ FOUND_ZERO
.
.
.
FOUND_ZERO:
; Do something...
```

The same with some instruction optimization:

```
XOR AX, AX
JZ FOUND_ZERO
.
.
.
FOUND_ZERO:
; Do something...
```

XOR instruction is a few bytes smaller than CMP, but XOR raises a CPU flag and so it works to compare to zero.
This saves a few CPU cycles. But with a good compiler and modern CPUs it becomes irrelevant.

Subject: Re: Socket Communication 101
Posted by nlneilson on Sat, 08 Dec 2012 15:48:26 GMT
View Forum Message <> Reply to Message

Thanks for the information.

I remember a Radio Shack from many years back. Very slow and the amount of memory was so small it hooked up to a cassette player to save bigger files.

I just got used to using \0 for end of transmission.

To be able to see as a string what is passed to be sent and what is output from the server was an advantage as nothing was done in the client or server except send and receive. With the C++ client in a header file and the Java server in it's own file it was easier to understand in different applications.

I looked at assembly language years ago but it was more than I wanted to get into.

I also got hung up on memory and saving space. External hdd are cheap now and recently picked up a 32GB micro SD card for $10 so spending too much time concerning memory is irrelevant also.

Times change and I am a bit slow keeping up and also hard headed to change my ways if an old way works.

Using U++ has been great to see what can be done and easier.

Subject: Re: Socket Communication 101
Posted by nejnadusho on Sun, 09 Dec 2012 04:11:05 GMT
View Forum Message <> Reply to Message

nlneilson, lectus,

Thank you very much to both of you, for the great help.

I just want to say that this thread is not 101 anymore.
It should be more like 101-102.

It became very in depth discussion which I am so happy I have been learning a lot from you.

Best,
Georgi

Subject: Re: Socket Communication 101
Posted by nlneilson on Fri, 14 Dec 2012 09:07:57 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 07 December 2012 11:14nlneilson wrote on Fri, 07 December 2012 13:45
Each sock.Put would be a different connect/disconnect, not good.

What makes you think that?

Mirek

Let me clarify what I meant.
Once a client and a server get connected that is fine and can stay connected even when nothing is being sent.

"Each sock.Put would be a different packet."
instead of connect/disconnect

AFAIR, going back to tinkering with sockets in Python, the amount of data sent in a large number of packets before errors is much less than the same amount of data sent in a single packet in the same amount of time.

Maybe something similar is uploading data to a server.
FileZilla does not have the ability to zip/unzip.
I tried uploading a directory/sub directories with several hundred files, took forever. The zip size is very close to the unzipped.
Zip it up and it went fairly fast and on the server I could unzip it with cpanel.

The biggest chunk I usually send is about 1GB and over 16,000 files. This of course is not a "socket" but it may be comparable to some extent.

I don't have any test data from before but I think the large packet was much faster than a large number of packets.

Maybe lectus has some information on this.

Subject: Re: Socket Communication 101
Posted by nlneilson on Sat, 15 Dec 2012 15:37:53 GMT
View Forum Message <> Reply to Message

This is a GPS GGA NEMA sentence:
$GPGGA,175226.000,3502.264328,N,11758.158283,W,2,8,,810.661, M,-31.8,M,,*73

This is stripped of everything except the latitude, longitude and altitude.
Then an ID such as a planes tail number and position number relative to a pilots aircraft.
35.030438,-118.167888,828.3,N10801,1
35.034438,-118.173888,1438.3,N10802,2
35.036438,-118.158888,1133.3,N10803,3
35.038438,-118.179888,1438.3,N10804,4
35.040438,-118.152888,1133.3,N10805,5
.....

To send this data with 50 lines each on individual packets with sock.Put(...) every second does

not work without errors.

Adding all 50 lines and stripping the \0 except at the end it can be sent in one packet every second without errors or problems.

Subject: Re: Socket Communication 101
Posted by nlneilson on Tue, 16 Apr 2013 01:54:32 GMT
View Forum Message <> Reply to Message

Spent some time optimizing code because of a sprained foot.
Some data was being transferred from a Upp app to a Java app through a socket as a character array.

Some survey paths were still being sent individually and Sleep(5); was necessary on a fast machine between each packet for each path or data would get corrupted or garbled. Sleep(10); was necessary on a slower machine.

Putting the data for 30 paths (60 lines) into a character array (about 2000 size) it was almost instantly transferred, no Sleep time necessary.

Subject: Re: Socket Communication 101
Posted by mirek on Tue, 16 Apr 2013 06:22:01 GMT
View Forum Message <> Reply to Message

That is kinda weird, my understanding is that kernel always buffers data before forming bigger packets, e.g.:

http://books.google.cz/books?id=ptSC4LpwGA0C&pg=PA58&amp
;lpg=PA58&dq=kernel+socket+buffers&source=bl&ots
=Kr6DLpdkNs&sig=oqyxevI71j7yqDxnrRhq4heArik&hl=en&am
p;sa=X&ei=qe1sUaaxJ4mNtQbn04H4Bg&sqi=2&ved=0CG8Q
6AEwCA#v=onepage&q=kernel%20socket%20buffers&f=false

Subject: Re: Socket Communication 101
Posted by nlneilson on Tue, 16 Apr 2013 17:51:32 GMT
View Forum Message <> Reply to Message

Thanks for the link, I saved that and also the document so it can be read off line.

Yes the data does get buffered but the difference was in my code.
Before it was like this:

getline

reformat
send

That was one packet.

getline
reformat
send

That was another packet.

etc. for each line.

It is the time required to send each individual packet that is important before they get corrupted or out of order.

Now the code is like this:

getline
reformat, keep the \n but remove the \0
add to buffer
get next line
reformat, keep the \n but remove the \0
add to buffer

etc until all lines are added to buffer
add \0
send

That is one packet

The amount of data or size of the buffer is irrelevant at least for the amount I have been working with.
By the time the Upp file choose box disappears all paths are displayed in the Java app.

Upp is great to be able to make changes.

Subject: Re: Socket Communication 101
Posted by mirek on Tue, 16 Apr 2013 18:06:27 GMT
View Forum Message <> Reply to Message

nlneilson wrote on Tue, 16 April 2013 13:51Thanks for the link, I saved that and also the document so it can be read off line.

Yes the data does get buffered but the difference was in my code.
Before it was like this:

getline
reformat
send

That was one packet.

getline
reformat
send

That was another packet.

Well, this is possible in case that your network is relatively faster than getline/reformat. See

http://en.wikipedia.org/wiki/Nagle%27s_algorithm

It still does not quite explain errors...

Mirek

Subject: Re: Socket Communication 101
Posted by nlneilson on Tue, 16 Apr 2013 20:43:45 GMT
View Forum Message <> Reply to Message

Thanks for the link.

The socket is just 'sending' data from the Upp app to a Java app.
There is no internet connection necessary but I do have a 100 MB connection.

The errors were shown up in the Java display app.

Here are a few lines from a file I open in Upp
// SW corner 35.028940,-117.970440,804.4 from S20 35.028928,-117.970460
35.04361500,-117.96188300
c,35.036235,-117.961830,35.036335,-117.961730

c,35.043615,-117.961883,35.028850,-117.961610 // correct ?? N 1/4 corner
c,35.043617,-117.962056,35.028850,-117.961610 // old N 1/4 corner

c,35.036291,-117.970649,35.036184,-117.952831

c,35.028940,-117.970440,35.028759,-117.952780
c,35.028759,-117.952780,35.043566,-117.952887
c,35.043663,-117.970880,35.043617,-117.962056

c,35.043617,-117.962056,35.043566,-117.952887
c,35.028940,-117.970440,35.043663,-117.970880 //W
// Farm
c,35.036291,-117.970649,35.036235,-117.961830
c,35.036235,-117.961830,35.037156,-117.961853
c,35.037156,-117.961853,35.037189,-117.966238
c,35.037189,-117.966238,35.039954,-117.966357
c,35.039982,-117.970770,35.039954,-117.966357

Each line that starts with c, is a path with two points and the other lines are ignored. Sometimes survey data is represented degrees, minutes and seconds rather than decimal degrees so the data is 'formatted' to decimal degrees and checked for like a latitude greater than 90 degrees, etc..

Most survey data is E->W or N->S that I am using.
Sending the data as a packet for each line without an adequate Sleep time many of the paths are diagonal as the points are out of order. Using print statements shows the data is not in order also.

Putting all the data in one buffer and sending in one packet all the data is received and displayed correctly.

Using shared memory between C++ and Java may be faster but was more complicated so a socket is being used.

This is displayed correctly, the two diagonal lines near the bottom is intended.
http://www.nlneilson.com/wwposts/Survey-Paths.png

The upper left is the Upp app.

This displays the errors:

http://www.nlneilson.com/wwposts/Survey-Paths-errors.png

Subject: Re: Socket Communication 101
Posted by mirek on Wed, 17 Apr 2013 06:03:56 GMT
View Forum Message <> Reply to Message

nlneilson wrote on Tue, 16 April 2013 16:43Thanks for the link.

The socket is just 'sending' data from the Upp app to a Java app.

OK, that explains packets (time for packet to pass from socket to socket is very low, so none get buffered).

Quote:
The errors were shown up in the Java display app.

That is still very weird. Frankly, given nature of TCP/IP protocol and the fact that U++ side of things is quite primitive, I would start looking for bug(s) in Java app...

Mirek

Subject: Re: Socket Communication 101
Posted by nlneilson on Sun, 05 May 2013 00:45:12 GMT
View Forum Message <> Reply to Message

Yes the problems seem to be on the java end.

Data is received but apparently there is no positive method to keep the data in order. Each 'line' represents two points. For the points to get messed up so they are not consistent with the path that is supposed to be drawn gives the diagonal lines.

The transfer is very fast whether sent separately or as one buffer. The time taken by the java app to change the format if necessary, error handling and draw it while additional lines are actually being received into some unknown que.

C++ and Upp are very good and most problems end up being in java.
