

Re: Fundamentals question, is this how it works?

Re: Fundamentals question, is this how it works?

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.networks/2006-08/msg00045>

- *From:* "Alexander Nickolov" <agnickolov@xxxxxxxx>
 - *Date:* Fri, 21 Jul 2006 14:15:35 -0700
-

Why don't you pick one thread and limit the discussion there?
I've answered you in like 4-5 threads already...

--

=====
Alexander Nickolov
Microsoft MVP [VC], MCS D
email: agnickolov@xxxxxxxx
MVP VC FAQ: <http://www.mvps.org/vcfaq>
=====

"Daniel" <DanielV@xxxxxxxxxxxxxxxxxxxx> wrote in message
news:ua1vjiOrGHA.2448@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Ahh that i already do and works fine. My problem lies in multiple packets
in a single stream. Knowin when that second packet started is my problem.
How do you find that?

"Alexander Nickolov" <agnickolov@xxxxxxxx> wrote in message
news:uqzAXfOrGHA.352@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

You maintain a buffer for the last incomplete packet. The next
time you receive data you try to finish the incomplete packet
instead of opening a new packet to receive. That's how you
process a TCP stream. Note it may take several reads to
complete a packet, and similarly you can get multiple packets
in a single read.

--

=====
Alexander Nickolov
Microsoft MVP [VC], MCS D
email: agnickolov@xxxxxxxx
MVP VC FAQ: <http://www.mvps.org/vcfaq>
=====

"Daniel" <DanielV@xxxxxxxxxxxxxxxxxxxx> wrote in message
news:OGCwa2OrGHA.3256@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Re: Fundamentals question, is this how it works?

How do you account for partial packet reassembly? Also i am using a stream, is it the same?

"Alexander Nickolov" <agnickolov@xxxxxxx> wrote in message
news:e1WaCuOrGHA.1976@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

You said you are using async socket, correct? Do not stop processing packets after you are done with one. The next may be there already. Only stop if you get WSAEWOULDBLOCK. Note the packet may be partially received when you get WSAEWOULDBLOCK so you have to account for partial packet reassembly in your code.

=====
Alexander Nickolov
Microsoft MVP [VC], MCSD
email: agnickolov@xxxxxxx
MVP VC FAQ: <http://www.mvps.org/vcfaq>
=====

"Daniel" <DanielV@xxxxxxxxxxxxxxxxxx>
wrote in message
news:OxymF6JrGHA.4236@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Hello and thanks for the reply.

That is what i thought i was saying that it receives it all in a stream receiving the buffer size each time. However could you explain the grouping to me some more?

If you mean this:

Actual bytes sent 234bytes
Receive buffer size 1024

On the receive end it reads

Re: Fundamentals question, is this how it works?

234 bytes into a 1024 byte buffer which is then read through, on reading the first 234 bytes you have your data.

If 234 bytes was sent then immediately after this 200 bytes was sent, then i would receive 234 bytes then immediately after this 200 bytes, so of my 1024 434 bytes of it have data in.

What i do is store the length of the data being sent in the first 4 bytes of the data being sent. So if 400 bytes are sent, then only 396 is actual data with the first 4 being a int for the length of data in bytes.

Then at the receive end i wait for data to come in and read the first 4 bytes, then continue to read that many bytes of the buffer data. On receiving that many bytes i then break and wait for the next set of data to come in. So in fact i am doing exactly what you said on i am assuming my data is coming in in groupings of my buffer size rather than taking it in on groupings of the actual data size sent, so if in my case i had 300 bytes come in then 400 on a 1024 byte buffer the second 400 would be ignored! I had put in a delay on data when it was

Re: Fundamentals question, is this how it works?

sent too quickly one after
the other and this fixed my
issues, i now
know why, because i was
merely forcing it to allow
time for empty
bytes in the stream so that
the next set of data would
start after the
first buffer was complete,
thus almost forcing it in
chunks!

I think i can finally remove
all these hack jobs i have
been doing now
and my confusions as to
whya delay was fixing my
issues is clear, i
understand now!!

One thing i am concerned
on, is it ok to do this then
say i read my
buffer and i find that out of
1024 bytes the first 10 bytes
are 0 then
the next byte is greater than
0 when looking at the values
in debug.
That would mean that the 0
has no data right? So when
looking for my
4bytes for the expected size
i can say to it if the byte=0
then skip,
and as soon as it hits a byte
which is greater than 0 then
read from
there to get the bytesize and
dont reset until u read all the
data in,
now repeat?

That acceptable or is there a
better more correct way to
check for
null data in a byte buffer?

Thanks!

Re: Fundamentals question, is this how it works?

"Michael K. O'Neill"

<MikeAThon2000@xxxxxxxxxxxxxxxxxxxx>

wrote in

message

news:%23UVCuTFRGHA.1140@xxxxxxxxxxxxxxxxxxxxxxxx

"Daniel"

<DanielV@xxxxxxxxxxxxxxxxxxxx>

wrote in

message

news:uDfBVwCrGHA.4864@xxxxxxxxxxxxxxxxxxxxxxxx

Hey

With

a

tcp

stream

socket

what

happens

when

it

is

reading

say

4000bytes

and

495

bytes

come

in?

I

am

finding

when

i

add

a

delay

in

my

reading

i

always

Re: Fundamentals question, is this how it works?

Re: Fundamentals question, is this how it works?

read
my
data
correctly.

If
i
let
it
run
at
full
speed
it
gets
stuck.
I
am
using
asynchornous
sockets.

On
my
understanding
is
this
right,
say
i
have
a
receive
buffer
of
1024

bytes

and
i
send
2048
bytes,
then
straight
after
i
send
400
bytes:

Re: Fundamentals question, is this how it works?

1)
Client
receives
1024
bytes
of
data
from
original
2048
sent,
and

processes

it
2)After
client
reads
all
the
data
it
receives
the
next
1024
bytes,
and
processes

it
3)
then
it
reads
the
first
400
bytes
and
processes
it.

Am
i
right
in
that
it
queues
up

Re: Fundamentals question, is this how it works?

Re: Fundamentals question, is this how it works?

that
way?

No, this is
not the way
it works.

TCP is a
stream-based
protocol,
which
means that
it ignores
any
attempt (on
the sending
side) to
somehow
group the
sent data
into
"messages"
or
"packets" or
whatever
you want to
call them. If
the sending
side
sends 2048
bytes, then
(assuming a
buffer large
enough) the
receiving
side might
get
all 2048
bytes in one
call to
recv(), or it
might need
many calls
to
recv()
before all
2048 bytes
are sent. If
the sending
side sends

Re: Fundamentals question, is this how it works?

2048 bytes
and
then sends
another 400
bytes, then
the
receiving
side might
get all
2448
bytes in one
call to
recv() or
(again) it
might need
many calls
to
recv()
before all
2448 bytes
are
received.
Moreover,
the
groupings
by which
these
bytes are
received
will
generally
ignore the
fact that
2048 bytes
were sent
first
followed by
400 bytes,
so a first
call to
recv() might
get 1531
bytes
(arbitrary
number, just
for
example's
sake), a
second call
to recv()
might

Re: Fundamentals question, is this how it works?

Re: Fundamentals question, is this how it works?

get 814
bytes, and a
third call
might get
the
remaining
103 bytes.

The only
thing that
TCP
guarantees
is that the
bytes will
be received
in the
correct
order. TCP
does not
guarantee
that the
bytes will
be
received in
the same
groupings
as when
they were
sent. In fact,
in a
degenerative
case
that
probably
would never
occur in
practice
(but is still
theoretically
possible),
only one
single byte
at a time
might be
retrieved by
each
call to
recv().

Re: Fundamentals question, is this how it works?

And
why
if
i
slow
down
the
reading
does
it
become
reliable
but
if
i
let
it
read
at
full
speed
it
gets
stuck.
When
i
say
stuck
i
mean
it
ignores
new
data
coming
in.
So
it
does
this,

but

this
is
accurate
to
my
real
situation

Re: Fundamentals question, is this how it works?

1)
Reads
425
bytes
2)
now
no
longer
reads
any
new
data

Is
it
possible
it
is
trying
to
read
too
quick?
since
it
is
in
a
loop

anyway

surely
it
has
to
finish
reading
the
first
before
it
can
continue
anyhow?

Re: Fundamentals question, is this how it works?

Generally, situations like this are caused by code that assumes that TCP preserves groupings. TCP itself is not stuck. However, because of the mistaken assumption, the code has encountered an unexpected situation, such as looking for a NULL terminator in c-style string, and then discarding everything after that. This is a mistake because TCP might have delivered part of the next string, such that the code discards valid data.

Re: Fundamentals question, is this how it works?