Example TCP sockets
(shown interactively)

Let's look at the "pattern" of the server:

1. Create a socket for TP
2. maybe set some "tuning" parameters
3. "bind" the socket to a port number
4. "listen" for a "connection" (sleep...zzz)

   Connect: a connection in TCP is the equivalent of a virtual link

5. {wakeup!} — someone wants to connect

   "accept" the new connection & create a new socket

6, 7, 8, ...

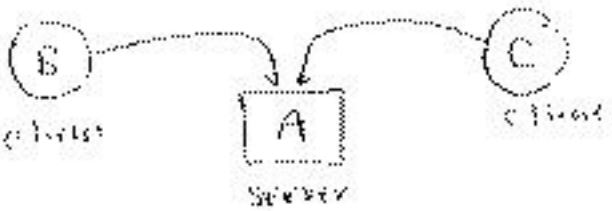
   send, receive, read, etc. —— then close when done.

Why does the TCP server pattern use two sockets?

— makes no sense when we think of a server with only one client at a time

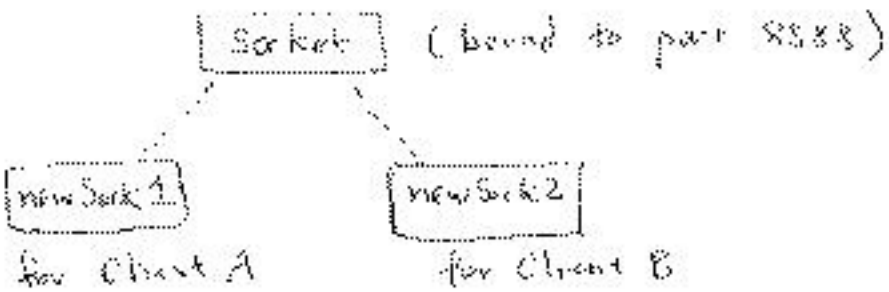
Answer:

Servers usually allow multiple clients to connect simultaneously.



both clients connect to the same port number!

Here's how the pattern would work



Unfortunately, this is difficult
to arrange!

Suppose Client A connects first...

Now, server has to listen (wait)
so that Client B can connect...

but what if Client A wants
some server response while the
server is waiting?

Two concurrency patterns help us
to find a solution to this puzzle

- Threads

  (each client gets a new thread)

- Select

  A Python module for event-driven
  network programming.

Important fact about TCP:

"writes" do not correspond to "reads"

This is because TCP is stream oriented.

EXAMPLE -

Server does

1. send "this is a message \n"
to a client

2. client when is waiting.

Client does

3. buff ~~[illegible]~~ sock.recv (1024)

print buff + "\n"

when gets output on console.

this is a me

notice: "ssage \n" is lost!

why?

The system puts TCP data into packets
in unpredictable ways

⇒ You need to plan for this!