# Lab Exercise 2: Simple Synchronous Sockets

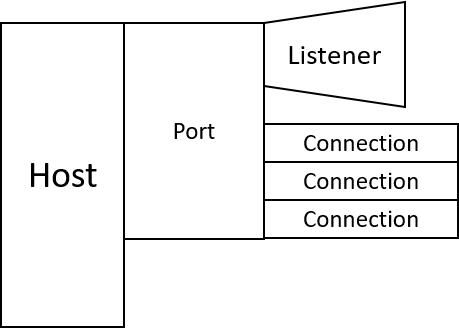
## Host and PortsSockets

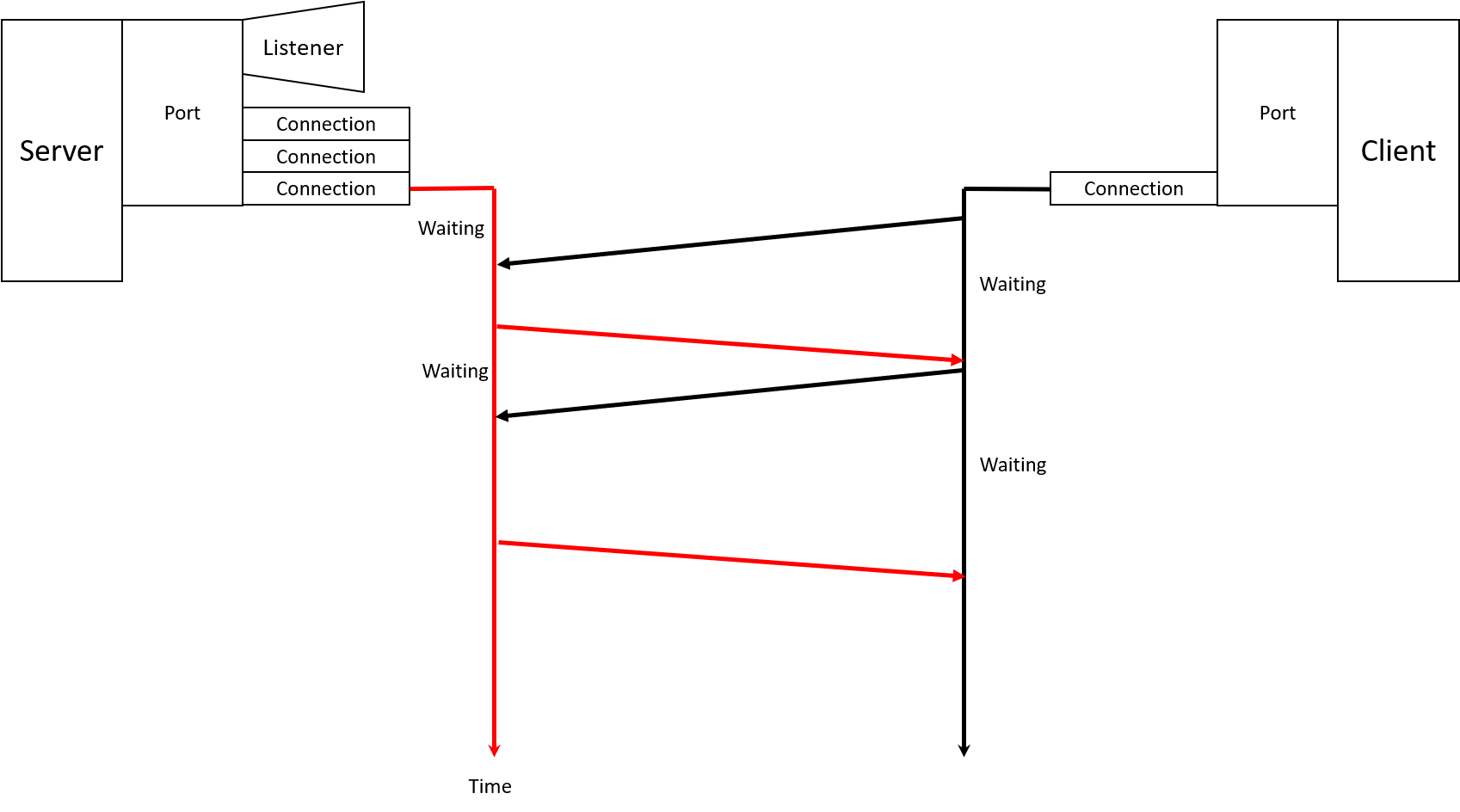
In this lab you are asked to undertake a reading and research exercise using Python sockets. The essential concepts we want to develop are an understanding of what a socket and connection are and how we can use these to transmit data between computers. From this we will develop an understanding of synchronous and asynchronous sockets -i.e. the ability to work on multiple things at once through threads. For this week we only really want to consider synchronous sockets.

|  |  |
| --- | --- |
|  | Why do we only have 65535 ports?  What does this tell us about socket design? |

In most modern computer systems we make use of TCP and UDP (or protocols based on these) as our two primary protocols for data transfer. The host platform provides us with a number of “ports” typically in the range 0-65535. Each of these ports can provide two-way communication between the host and multiple remote end-points.

Ports below 1024 are known as the “well-known” ports and are typically reserved for system processes to provide network services. You shouldn’t generally utilise anything in this range. Above this between 1024 and 49151 we have sockets that have a registered (through IANA) usage – you can utilise these however it is wise not to do so as you are potentially conflicting with other services your user may use. Finally from 49152 – 65535 the ports are free for your usage for projects.

Each of these sockets can have a single listener that waits for new connections to be established. Typically this is used by a single process however there are mechanisms to allow multiple processes to share access to a single socket. When an incoming connection request is accepted a connection is established which uniquely identifies the connection using a 4-tuple of “local IP:Local Port:Remote IP:Remote Port”. This allows multiple connections to be managed on a single port as each is uniquely identifiable.  
  
The idea of a synchronous connection is one in which we must wait for the response from the other side or wait on something (an IO device, a resource becoming free etc.). This is generally a bad idea in most networked programs as we could be doing something useful while we wait rather than just waiting however it is the simplest way to introduce the concept of networked applications. We will look at threading and asynchronous networking next week to make these programs more useful. Think of synchronous networking as being stuck waiting for a package – you know it will arrive at some point in the next 5 minutes – 5 hours… but you can’t leave the house or you will miss it.



1. Real Python has a great tutorial to get us started on this week’s work. You should read through the exercises down to the start of Multi-connection client and server (we will be looking at multiple connections next week). <https://realpython.com/python-sockets/>
2. From this example we have seen communication that sends data once and then quits the application, consider and implement a solution which allows for data to be sent repeatedly until the user gives an exit command (for the purposes of this exercise we will use \*#0#\* to represent our exit command as it is unlikely to be a non-deliberate string).
3. At this point your tutor will discuss the concept of state machines with you. Python has a state machine library now that you will want to look at the following reading:

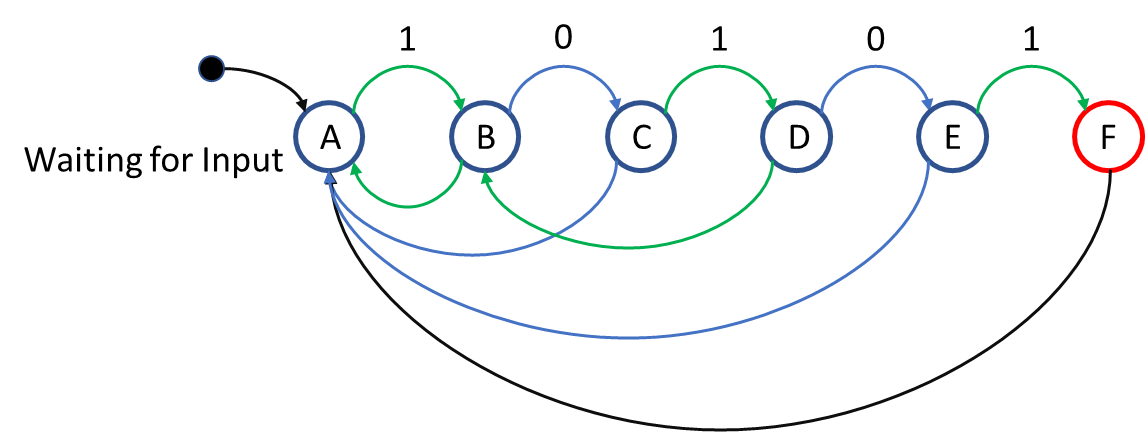
|  |  |
| --- | --- |
|  | <https://buildmedia.readthedocs.org/media/pdf/python-statemachine/stable/python-statemachine.pdf>  <https://www.python-course.eu/finite_state_machine.php>  <https://medium.com/@brianray_7981/tutorial-write-a-finite-state-machine-to-parse-a-custom-language-in-pure-python-1c11ade9bd43> |

State Machines

(<https://en.wikipedia.org/wiki/Finite-state_machine>)

Consider a system to track digit input – we are looking for a 5 digit input like a pin code: We could do a sliding window input and look at each 5 digit section of input, or do something more complex like a regex. In this way we have to check each piece of the input against all of the values – we maintain a history of where we are, but not what we have done.

The acceptable string (green)

A state machine by contrast maintains some concept of what we have done, and where we are – but only in so far as it is relevant to our functionality. So if we are looking for input we can just keep track of where we should be in the sequence and not what came before.

So considering the state machine above – we could represent this in python as:

1. while True:  
    inputvar = input("Pleae enter a digit")  
    if currentstate == "W":  
    DoStateW()  
    previousstate = currentstate  
      
    if inputvar == "1":  
    currentstate = "A"  
    else:  
    currentstate = "W"  
    elif currentstate == "A":  
    DoStateA()  
    previousstate = currentstate  
     
    if inputvar == "0":  
    currentstate = "B"  
    else:  
    currentstate = "W"

This resembles the switch statements in C style languages – we could of course exploit Pythonesque structures to do this as a dictionary (<https://data-flair.training/blogs/python-switch-case/>), or use the State Machine class (<https://python-3-patterns-idioms-test.readthedocs.io/en/latest/StateMachine.html>) that exists in Python 3.

Modifying a state machine is easy – we can add new states and transitions easily and only define their linkages, we don’t have to alter the check code or regex that’s implementing things for us. The power of state machines is really expressed well in systems like SMTP when we have a subset of inputs that we can allow at any given time and we want to control these – we reduce our tests and complexity by implementing state.

|  |  |
| --- | --- |
|  | Redraw the state machine to handle a 6th digit.  How many changes did you need to make to your diagram and code? |

Reading Links:

|  |  |
| --- | --- |
|  | <https://www.smashingmagazine.com/2018/01/rise-state-machines/>  <https://blog.markshead.com/869/state-machines-computer-science/>  <https://www.youtube.com/watch?v=lffJJbqt1fk>  <https://www.youtube.com/watch?v=-Yicg2TTMPs> |

1. Once you have the program able to handle repeatedly sending data and the exit condition implement the number guessing game. The key to this exercise is that you will need to parse out what the user is sending you – whether it is a guess or a command. Your server will need to maintain some form of state to handle returning the number of guesses.
2. Extend this exercise to Hangman instead of the number guessing game, at this point you likely want to consider adding some state on both the client side and the server side rather than simply on the server side – why even send a guess if you know it can’t be valid.