Computer Networking Coursework Assignment:

James Cordery – 13184228

1st May 2020

The purpose of this assignment is to create client and server programmes in the UDP format that pass messages between them and print the results.

Client:

For the client programme, a socket is set up and the user is prompted to enter 3 strings, *S1*, *S2* and *S3*.

These 3 strings are then sent to a server in three distinct UDP packets.

The client then waits for a string *R* and an integer *n* from the server.

Finally, the client removes the *'#'* from the received message and checks that the concatenation and length match the original entered strings and prints the results.

Server:

For the server programme, a server socket is set up and the server waits for a series of 3 UPD packets to come through from the client.

The server then concatenates the 3 packets, separating them with two *'#'* characters and computes them as *R*. It also adds the three packet lengths together and computes them as *n*.

Finally, the server sends *R* and *n* to the client as two UDP packets.

Coding:

I started by creating the client programme. The first thing I had to do was import the socket and socketserver modules. Then, I created a socket by which the UDP packets could be transmitted as well as define the Host, IP and Port address. I chose local host so that it works within the computer being used and picked a random port number that was above 1000 as the lower ones are reserved. As the UDP is being used, there is no need to establish a secure connection before transmitting data, making it quicker although it is less secure.

To create the three distinct UDP packets I simply created three strings that prompt the user to enter three separate pieces of data. In this case, a word broken down into 3 segments. As each packet was entered, I encoded them to turn them into a bytes package ready to be transmitted and then sent each one to the server, one after another. I then printed a message to inform the user of the complete message by concatenating the 3 individual packets as well as printing a message to inform the user of the total message length being sent. The client programme then waits for a return message from the server.

I then created the server in the same way. I created the socket and bound the server to it. To get the server to wait I created a while loop that set the server to listen for three distinct packets from a client. It was set to wait until all three were received before executing the next step.

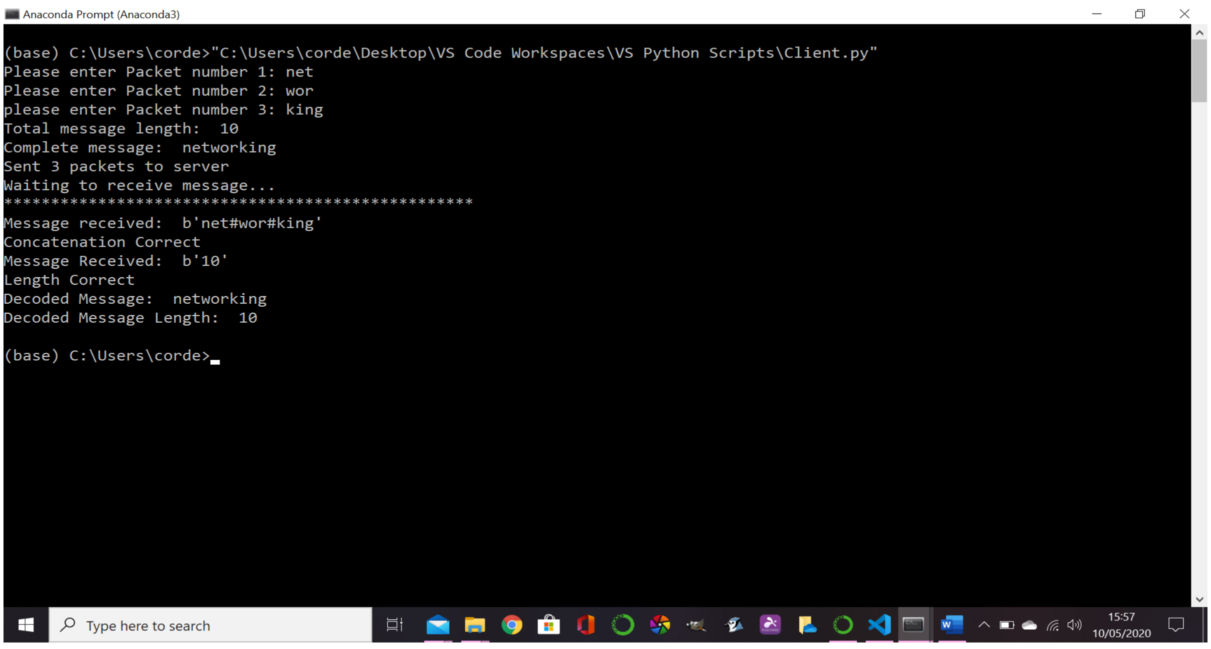
Once it had received the three packets, I set it to decode the messages using the .decode() function as they were sent as bytes. This converted them back to strings. Once this was done, I added the lengths of the 3 packets together and printed the result as n. I also concatenated the 3 packets, adding two '#' separators and printed the complete word as R. Finally, I encoded n and R into bytes and sent them as two distinct packets back to the client.

Lastly, back on the client side, it awaits the two packets. Once it receives the concatenated packet R, it decodes it back into a string and removes the separators using the .replace() function. It then checks that the word then matches the original joined packets using an If statement. If they match then it prints a successful message, otherwise it prints an error. It also waits for packet n and converts it back into an integer. Again, an If statement is run to check that the total word length sent form the server matches the original total and prints either a successful message if its correct or an error. Finally once this has been completed, the client socket closes.

Below are two executions of the client/server functions. The first is a standard run to show that the programme works. The second execution I set up to fail to make sure that the If statement checks work correctly.

<u>First Execution:</u>

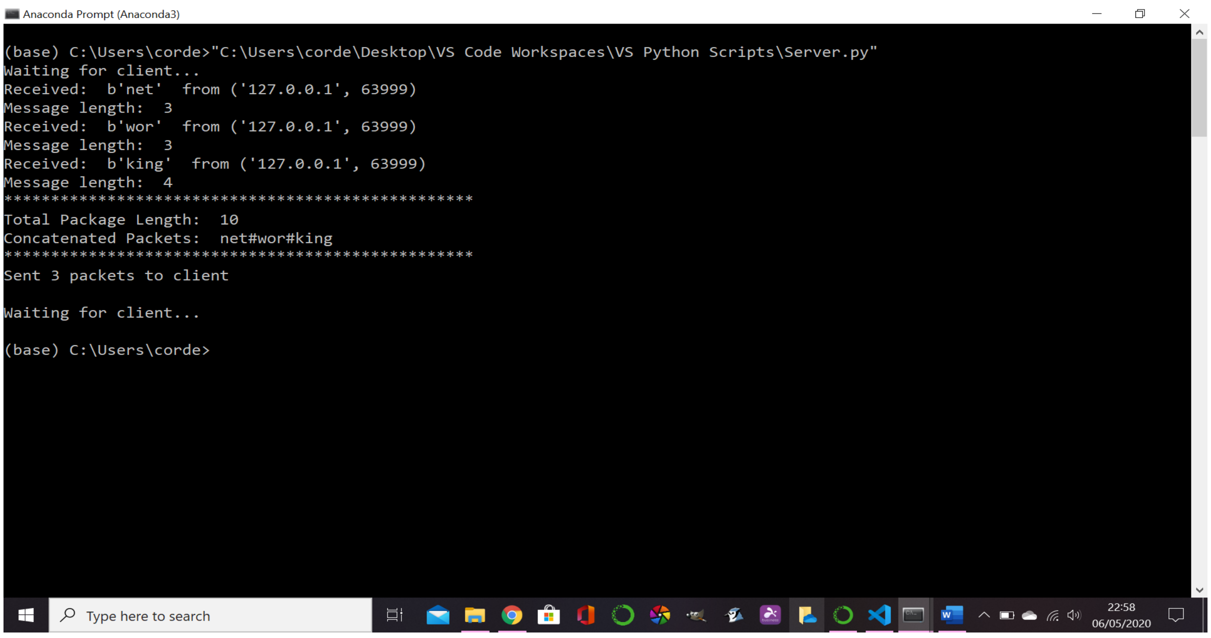
1.) This is the client programme. Here I have entered the first packet (*s1*) as net, the second packet (*s2*) as wor and the third packet (*s3*) as king. The complete word is networking and the message length is 10. These packets are then transmitted to the server and the client waits for a response.
2.) The second part of this shows the message received back from the server. It is the concatenated word with two *'#'* separators added. It then removes the separators and checks that the concatenated word and the length matches the original joint packets and prints messages to confirm this or advise that they don't match and there is an error. Finally, it prints the complete word and the total length before closing the socket.


```
(base) C:\Users\corde>"C:\Users\corde\Desktop\VS Code Workspaces\VS Python Scripts\Client.py"
Please enter Packet number 1: net
Please enter Packet number 2: wor
please enter Packet number 3: king
Total message length:  10
Complete message:  networking
Sent 3 packets to server
Waiting to receive message...
**************************************************
Message received:  b'net#wor#king'
Concatenation Correct
Message Received:  b'10'
Length Correct
Decoded Message:  networking
Decoded Message Length:  10

(base) C:\Users\corde>_
```

1. On the server side, it starts up and waits for a series of messages from the client. It then receives 3 distinct packets from the client, displaying the message as well as the IP and port that it was sent from.

2. In the second part, it has added the two '#' separators and added the packets together to make *net#wor#king*. It prints the complete word and word length and then sends these back to the client as two UDP packets.



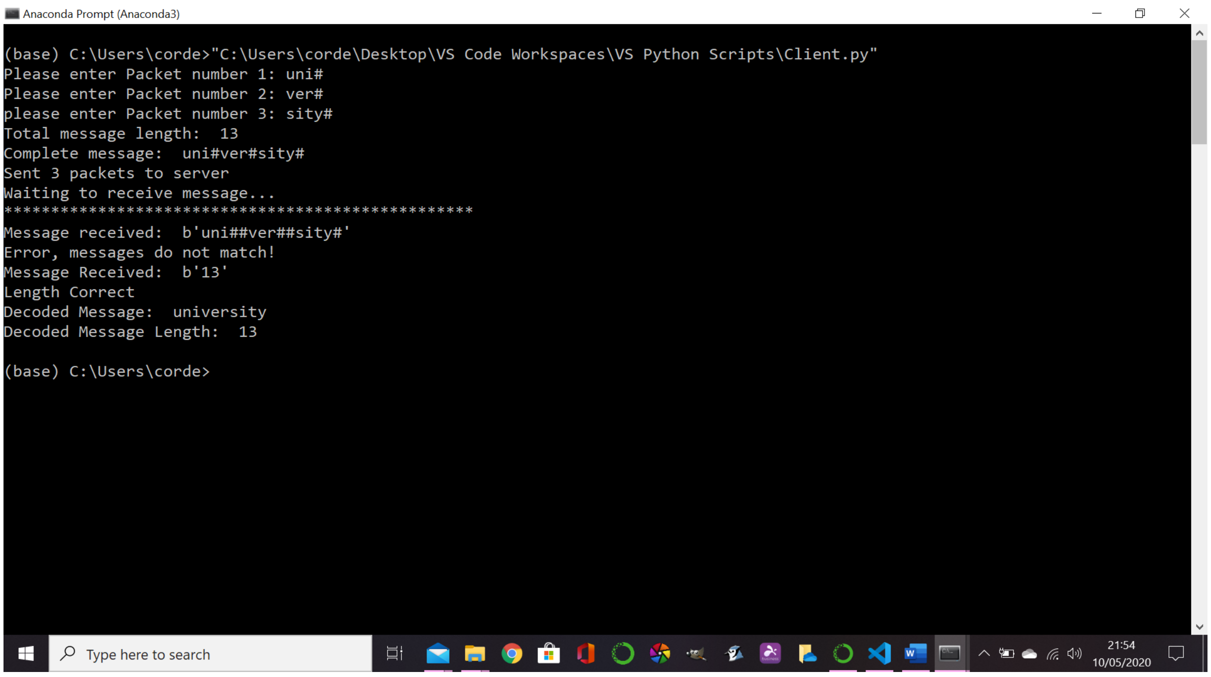
```
(base) C:\Users\corde>"C:\Users\corde\Desktop\VS Code Workspaces\VS Python Scripts\Server.py"
Waiting for client...
Received:  b'net'  from ('127.0.0.1', 63999)
Message length:  3
Received:  b'wor'  from ('127.0.0.1', 63999)
Message length:  3
Received:  b'king'  from ('127.0.0.1', 63999)
Message length:  4
**************************************************
Total Package Length:  10
Concatenated Packets:  net#wor#king
**************************************************
Sent 3 packets to client

Waiting for client...

(base) C:\Users\corde>
```

Second Execution:

1. In this second execution, I again entered three packets, but this time also added a *'#'* at the end of each one. The client still sends each message to the server, prints the length and concatenated word and then awaits a return message.
2. In the second phase however, it receives the returned word and length but because I added the extra '#'character, the check fails as the server side adds them as separators as well, so the word comes back with double the amount. This flags up an error message as the word does not match the original joined packets. However, the check still runs to remove the *'#'* separators added by the server, but it removes all the *'#'* characters including the ones I added so the final word prints but does not match what was originally sent. This could be rectified by adding a message advising the user not to use those characters.

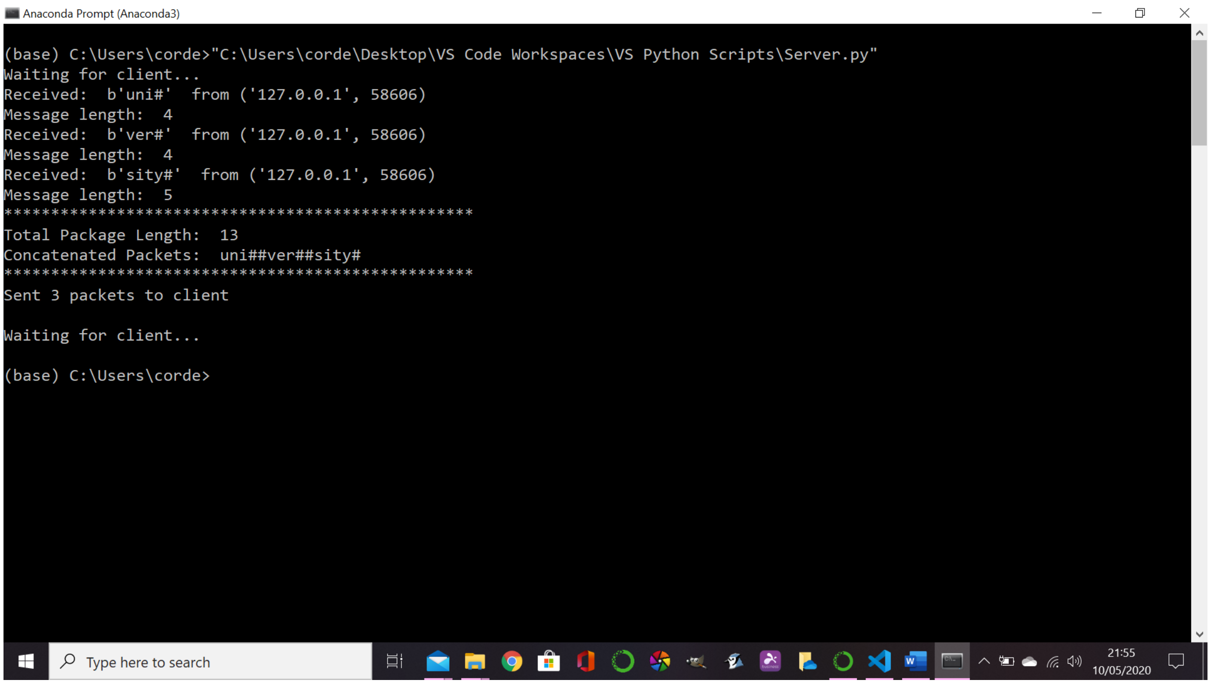


Anaconda Prompt (Anaconda3)

```
(base) C:\Users\corde>"C:\Users\corde\Desktop\VS Code Workspaces\VS Python Scripts\Client.py"
Please enter Packet number 1: uni#
Please enter Packet number 2: ver#
please enter Packet number 3: sity#
Total message length:  13
Complete message:  uni#ver#sity#
Sent 3 packets to server
Waiting to receive message...
**************************************************
Message received:  b'uni##ver##sity#'
Error, messages do not match!
Message Received:  b'13'
Length Correct
Decoded Message:  university
Decoded Message Length:  13

(base) C:\Users\corde>
```

1. On the server side we can see that it waits for the client as before and receives the three distinct UDP packets. It displays the packets, their individual length and the server and port it received them from.
2. It then prints the total message length and adds the two '#' separators and concatenates the packets. Because I added a '#' character to the end of each packet the complete word spells *uni##ver##sity#.* It then sends the message length and concatenated packet back to the client.



Anaconda Prompt (Anaconda3)

```
(base) C:\Users\corde>"C:\Users\corde\Desktop\VS Code Workspaces\VS Python Scripts\Server.py"
Waiting for client...
Received:  b'uni#'  from ('127.0.0.1', 58606)
Message length:  4
Received:  b'ver#'  from ('127.0.0.1', 58606)
Message length:  4
Received:  b'sity#'  from ('127.0.0.1', 58606)
Message length:  5
**************************************************
Total Package Length:  13
Concatenated Packets:  uni##ver##sity#
**************************************************
Sent 3 packets to client

Waiting for client...

(base) C:\Users\corde>
```