Name: MD: Aynul Islam

Chinese Name: 叶子

Student Id: 4420190030

Sockets with Python buffering and streaming data

There are a few logical ways that you could handle for this, but one common way is by starting all messages with a header that contains the length of the message that is going to come. The next challenge is normalizing this header in some way. You might consider using some series of characters, or some format, but then you run the risk of people accidentally, or purposefully, mimicking this formatting. Instead, you can go with a fixed-length header, where the first n bytes of data will be the header data, which will include the length of the message to come. Once we've received that length of data, we know any following information will be a new message, where we need to grab the header and continue repeating this process.

```
#Aligning the text and specifying a width:

>>>
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill char
'***********centered***********'
```

this case, you can see various examples where there are 30 characters used every time, but you can do various alignments. While this is mainly used to make text-based GUIs pretty, we can also use this for our purposes, like:

```
f'{len("your message here!"):<10}'
```

In the above case, this will produce the length of our message using 10 characters.

```
>>> f'{len("your message here!"):<10}''18
```

server.py:

```
import socket

HEADERSIZE = 10

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((socket.gethostname(), 1241))
s.listen(5)

while True:
    # now our endpoint knows about the OTHER endpoint.
    clientsocket, address = s.accept()
    print(f"Connection from {address} has been established.")

    msg = "Welcome to the server!"
    msg = f"{len(msg):<{HEADERSIZE}}"+msg

    clientsocket.send(bytes(msg,"utf-8"))
```

So now our messages will have a header of 10 characters/bytes that will contain the length of the message, which our client use to inform it when the end of the message is received. Let's work on the client.py next:

client.py

```
import socket

HEADERSIZE = 10

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((socket.gethostname(), 1241))

while True:
    full_msg = ''
    new_msg = True
    while True:
        msg = s.recv(16)
        if new_msg:
            print("new msg len:",msg[:HEADERSIZE])
            msglen = int(msg[:HEADERSIZE])
            new_msg = False

        print(f"full message length: {msglen}")

        full_msg += msg.decode("utf-8")

        print(len(full_msg))


        if len(full_msg)-HEADERSIZE == msglen:
            print("full msg recvd")
            print(full_msg[HEADERSIZE:])
            new_msg = True
```

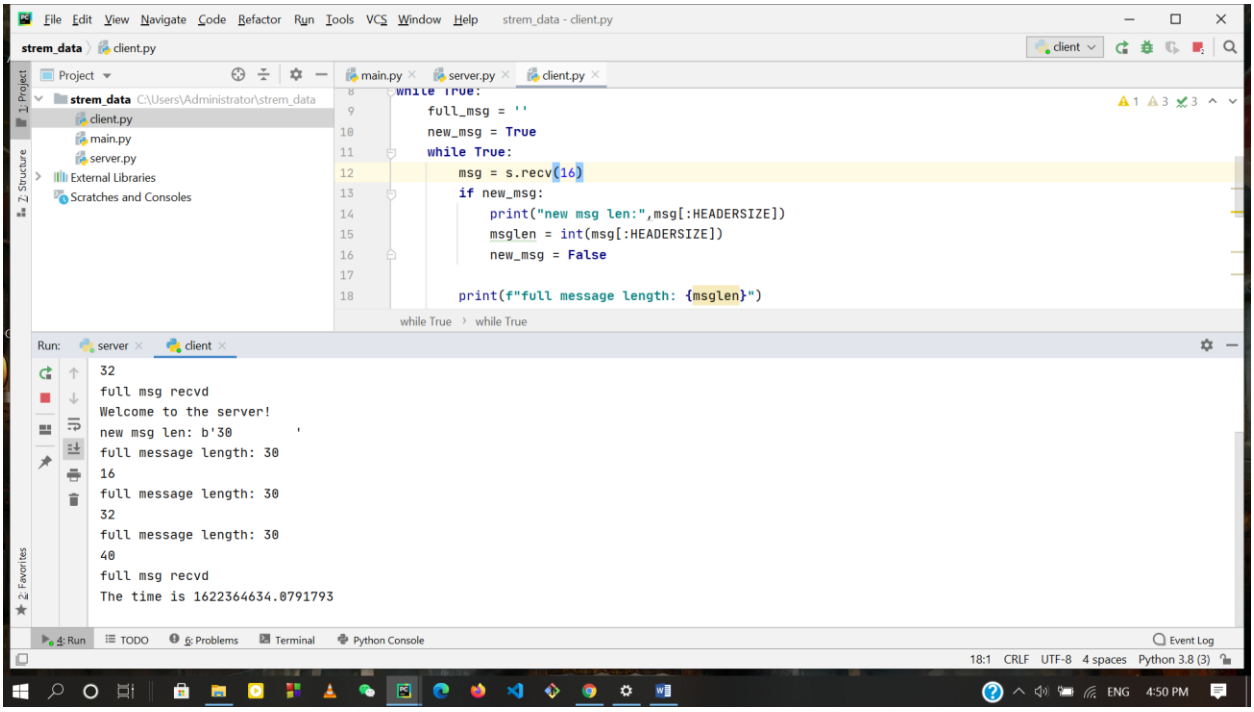
This one is a bit more involved, but nothing too crazy here. I increased out buffer to 16 bytes. 8 wouldnt even be enough to read the header, so that would have been a problem, and you would probably never have a buffer as small as these anyway. We're just doing it for example. So, we start off in a state where the next bit of data we get is a new_msg.

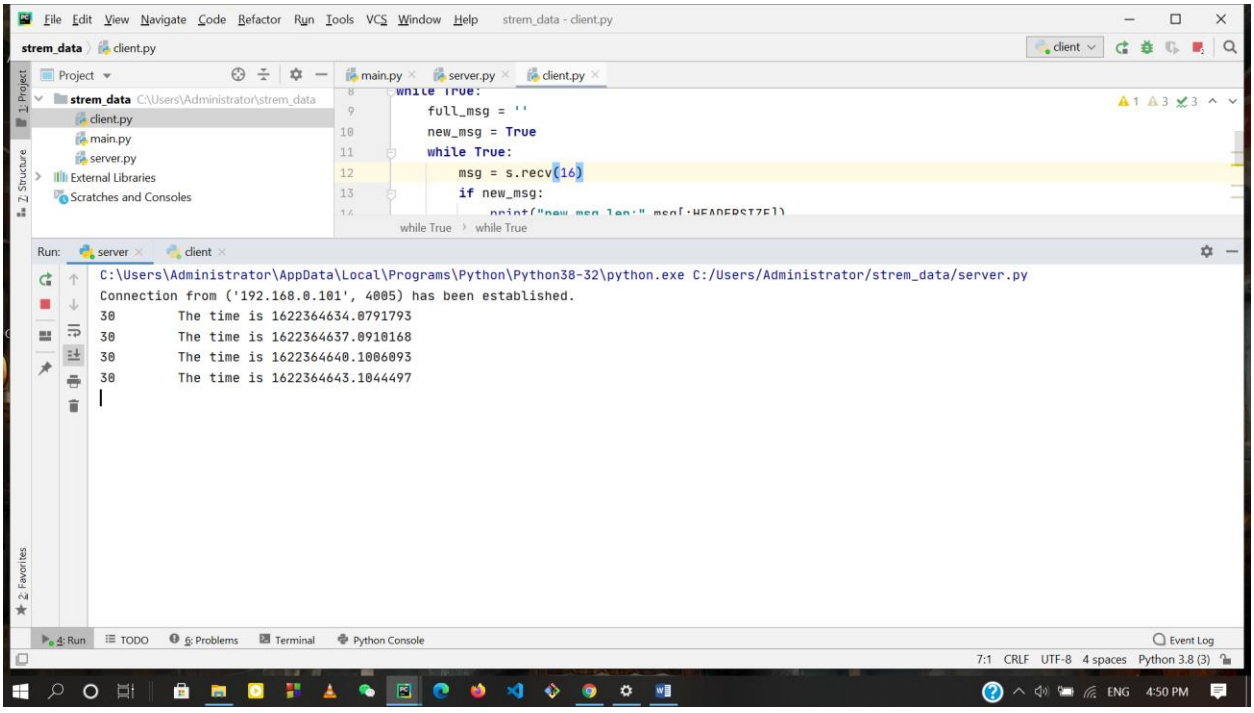
we just add the following to the end:

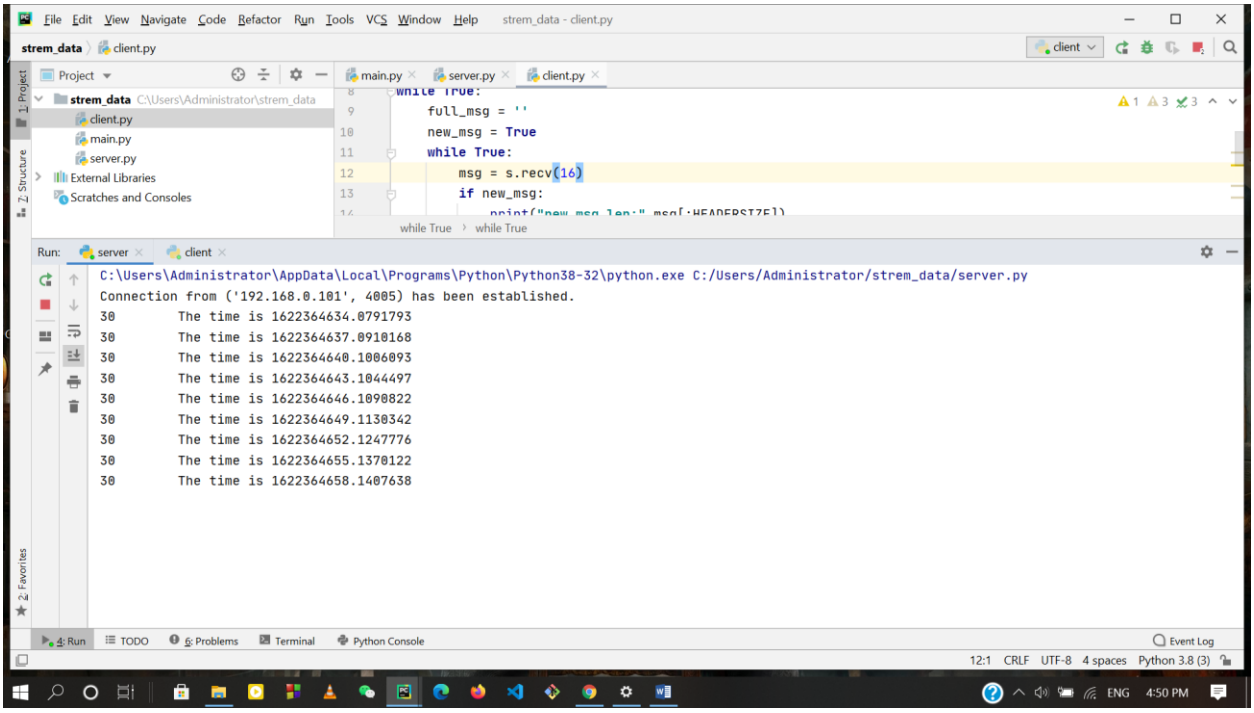
```
    while True:
        time.sleep(3)
        msg = f"The time is {time.time()}"
        msg = f"{len(msg):<{HEADERSIZE}}"+msg

        print(msg)
```

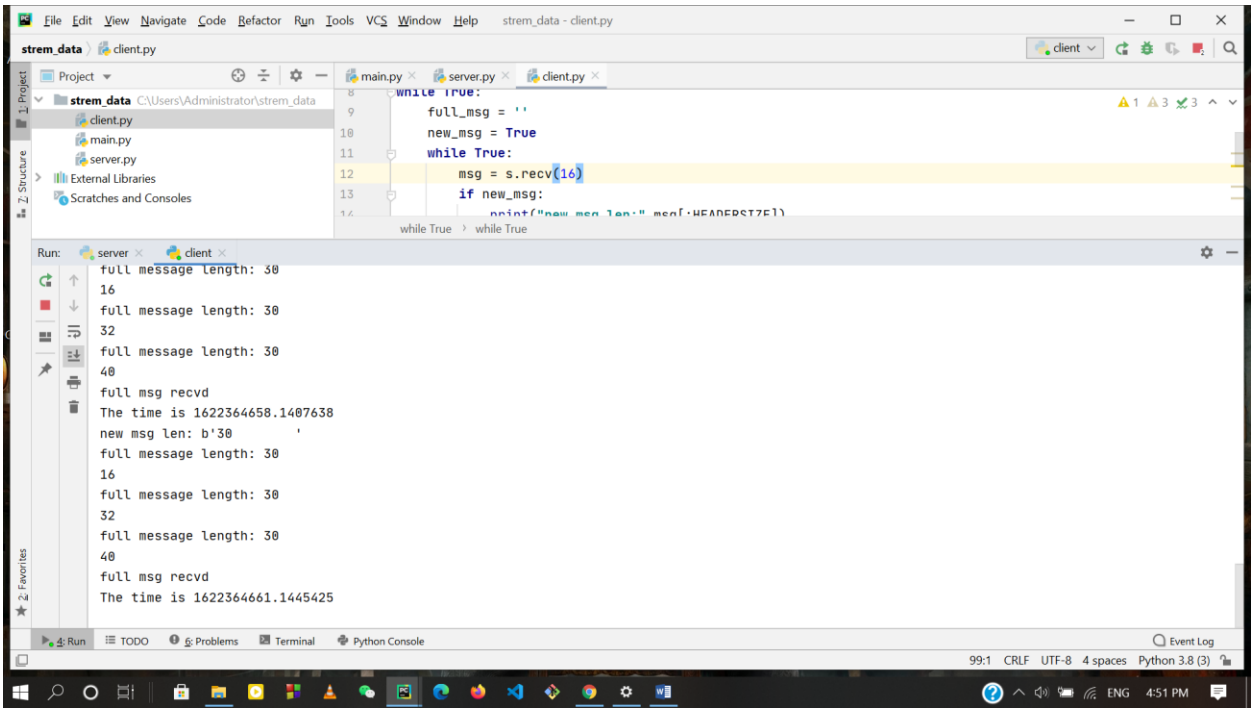
```
        clientsocket.send(bytes(msg,"utf-8"))
```

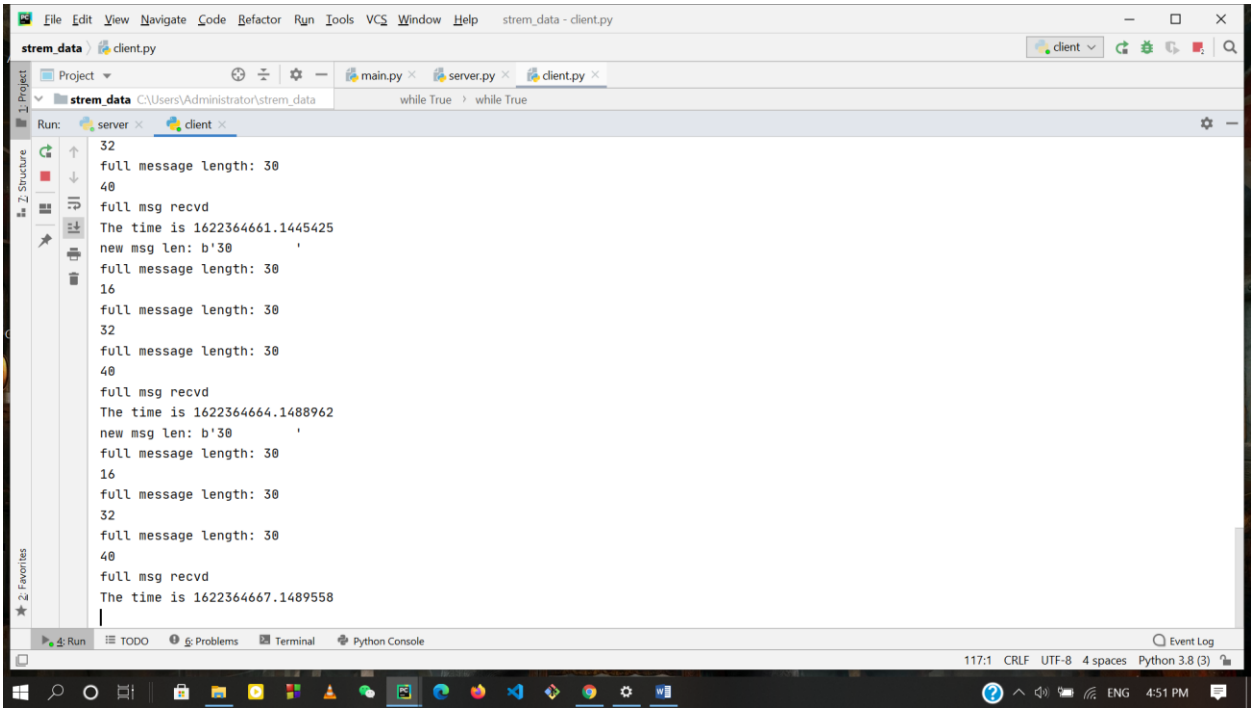
program screenshot :






File Edit View Navigate Code Refactor Run Tools VCS Window Help strem_data - client.py
strem_data client.py
client
Project
strem_data C:\Users\Administrator\strem_data
client.py
main.py
server.py
External Libraries
Scratches and Consoles
main.py server.py client.py
while True:
full_msg = ''
new_msg = True
while True:
msg = s.recv(16)
if new_msg:
while True › while True
Run: server client
C:\Users\Administrator\AppData\Local\Programs\Python\Python38-32\python.exe C:/Users/Administrator/strem_data/server.py
Connection from ('192.168.0.101', 4005) has been established.
30 The time is 1622364634.0791793
30 The time is 1622364637.0910168
30 The time is 1622364640.1006093
30 The time is 1622364643.1044497
30 The time is 1622364646.1090822
30 The time is 1622364649.1130342
30 The time is 1622364652.1247776
30 The time is 1622364655.1370122
30 The time is 1622364658.1407638
4: Run TODO 6: Problems Terminal Python Console Event Log
12:1 CRLF UTF-8 4 spaces Python 3.8 (3)
ENG 4:50 PM


File Edit View Navigate Code Refactor Run Tools VCS Window Help strem_data - client.py
strem_data client.py
client
Project
strem_data C:\Users\Administrator\strem_data
client.py
main.py
server.py
External Libraries
Scratches and Consoles
main.py server.py client.py
while True:
full_msg = ''
new_msg = True
while True:
msg = s.recv(16)
if new_msg:
while True › while True
Run: server client
full message length: 30
16
full message length: 30
32
full message length: 30
40
full msg recvd
The time is 1622364658.1407638
new msg len: b'30 '
full message length: 30
16
full message length: 30
32
full message length: 30
40
full msg recvd
The time is 1622364661.1445425
4: Run TODO 6: Problems Terminal Python Console Event Log
99:1 CRLF UTF-8 4 spaces Python 3.8 (3)
ENG 4:51 PM


File Edit View Navigate Code Refactor Run Tools VCS Window Help strem_data - client.py
strem_data client.py
client
Project
main.py server.py client.py
strem_data C:\Users\Administrator\strem_data
while True while True
Run: server client
32
full message length: 30
40
full msg recvd
The time is 1622364661.1445425
new msg len: b'30 '
full message length: 30
16
full message length: 30
32
full message length: 30
40
full msg recvd
The time is 1622364664.1488962
new msg len: b'30 '
full message length: 30
16
full message length: 30
32
full message length: 30
40
full msg recvd
The time is 1622364667.1489558
Run TODO Problems Terminal Python Console
Event Log
117:1 CRLF UTF-8 4 spaces Python 3.8 (3)
ENG 4:51 PM