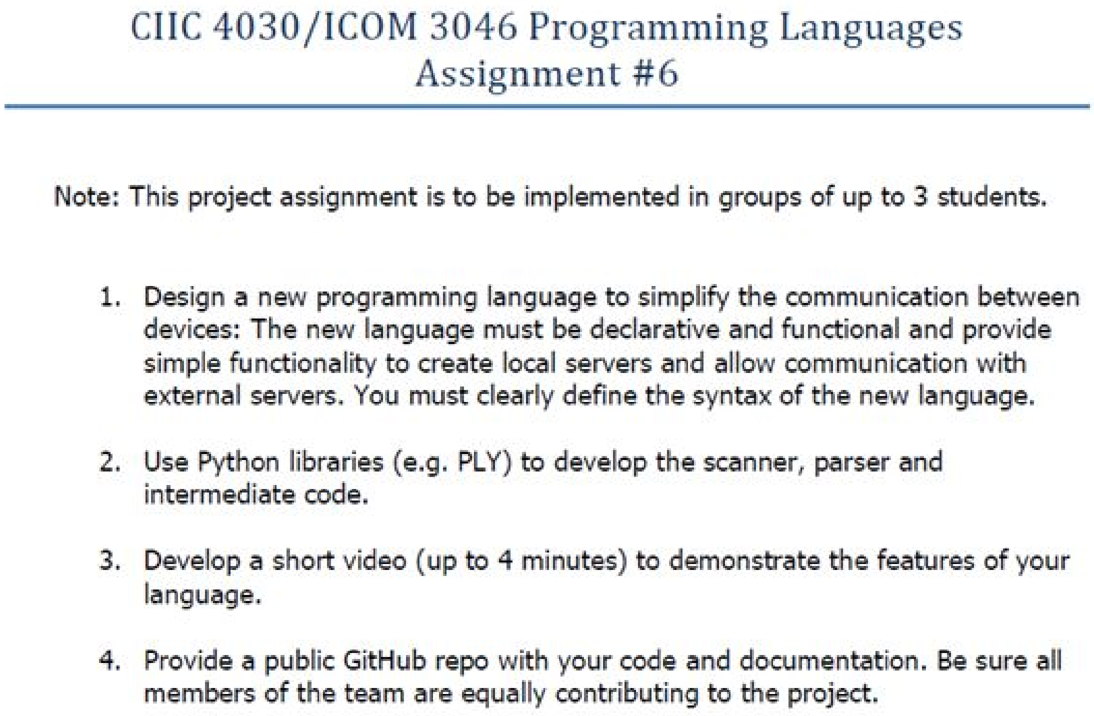
CIIC4030/ICOM3046 Programming Language Assignment#6

Documentation

Edgar J. Rivera Hernandez

Edwin X. Borrero

Fabian O. Guzmán



CIIC 4030/ICOM 3046 Programming Languages
Assignment #6

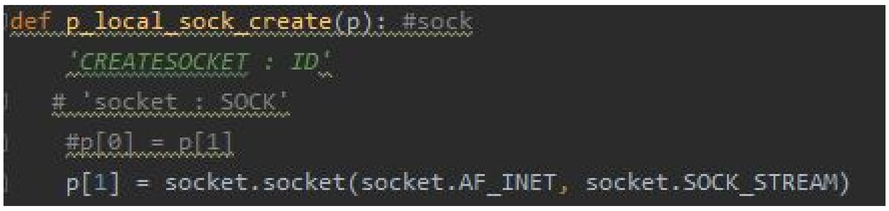
Note: This project assignment is to be implemented in groups of up to 3 students.

1. Design a new programming language to simplify the communication between devices: The new language must be declarative and functional and provide simple functionality to create local servers and allow communication with external servers. You must clearly define the syntax of the new language.
2. Use Python libraries (e.g. PLY) to develop the scanner, parser and intermediate code.
3. Develop a short video (up to 4 minutes) to demonstrate the features of your language.
4. Provide a public GitHub repo with your code and documentation. Be sure all members of the team are equally contributing to the project.

In this project we made a parser implementation that facilitates the "coding" of both local and external server. The Local Server is in TCP using localhost and the external server is between two machines (or in this computers). Although it has to be said that the external server only works if the client knows the local internet IP address of the "server" machine. In the Git you can see that also added some simple server/client classes. For example **socket_echo_client** contains code of a local client that sends repeating messages to the local server. The class **socket_echo_server** contains code for a local server that can receive multiple messages. The class **socket_external_client** contains code for an external "client" that connects to a machine connected to **RUMNET**, it also lets the "client" device send multiple messages to the "server" device. The class **socket_external_server** contains the code for an external server that can receive multiple messages from a client device. The class **lex.py** contains the "scanner" for the language and **parser.py** is the parser code that also contains the intermediate code. The reason that for the external client be connected to a fixed

IP address is that we were unable to figure out how can the external client can connect to the external server without knowing the server`s IP Address.

We are going to begin by describing the parser`s methods and definitions:

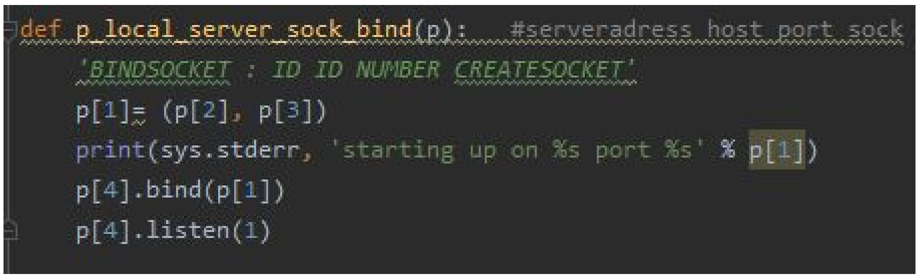


```
def p_local_sock_create(p): #sock
    'CREATESOCKET : ID'
   # 'socket : SOCK'
    #p[0] = p[1]
    p[1] = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

This code defines **CREATESOCKET** as an ID which refers to the “name” of a socket that can be used for the local server or local client.

This refers to the code line: sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

In general, this definition represents a creation of a socket.



```
def p_local_server_sock_bind(p):   #serveradress host port sock
    'BINDSOCKET : ID ID NUMBER CREATESOCKET'
    p[1]= (p[2], p[3])
    print(sys.stderr, 'starting up on %s port %s' % p[1])
    p[4].bind(p[1])
    p[4].listen(1)
```

This code defines **BINDSOCKET** as an ID ID NUMBER CREATESOCKET which refers to variables named ServerAddress, Host, Port, Socket.

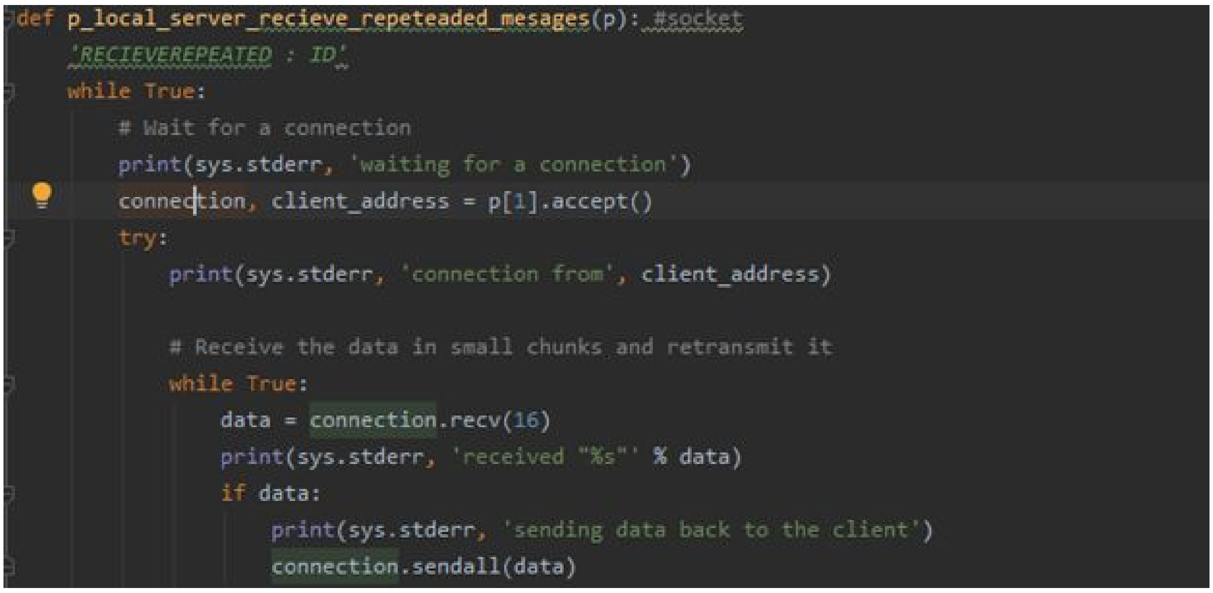
In general this definition binds the socket to the port, assigns the local address to “localhost” and Port, and make the socket listen for connections.

This definition is equivalent to the following lines of code:

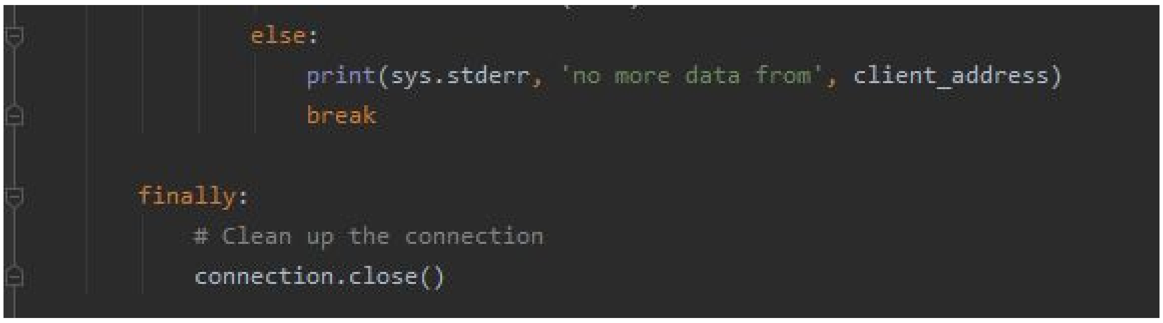
server_address = (‘localhost’, Port)

socket.bind(server_address)

socket.listen(1)



```
def p_local_server_recieve_repeteaded_mesages(p): #socket
    'RECIEVEREPEATED : ID'
    while True:
        # Wait for a connection
        print(sys.stderr, 'waiting for a connection')
        connection, client_address = p[1].accept()
        try:
            print(sys.stderr, 'connection from', client_address)

            # Receive the data in small chunks and retransmit it
            while True:
                data = connection.recv(16)
                print(sys.stderr, 'received "%s"' % data)
                if data:
                    print(sys.stderr, 'sending data back to the client')
                    connection.sendall(data)
```



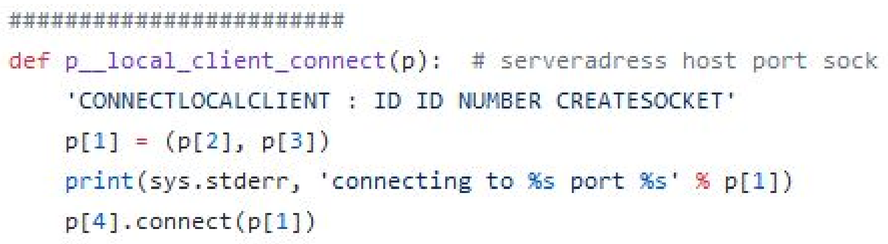
```
                else:
                    print(sys.stderr, 'no more data from', client_address)
                    break

        finally:
            # Clean up the connection
            connection.close()
```

This code defines **RECIEVEREPEATED** as an ID, which refer to a socket variable name

In general, this definition allows the local server socket to receive repeated messages from a client.

The only part were the socket is used here is the line:

Client_adress = sock.accept();



```
########################
def p__local_client_connect(p):  # serveradress host port sock
    'CONNECTLOCALCLIENT : ID ID NUMBER CREATESOCKET'
    p[1] = (p[2], p[3])
    print(sys.stderr, 'connecting to %s port %s' % p[1])
    p[4].connect(p[1])
```

This code defines **CONNECTLOCALCLIENT** as an ID ID NUMBER CREATESOCKET which refer to the variables named ServerAddress Host Port Socket.

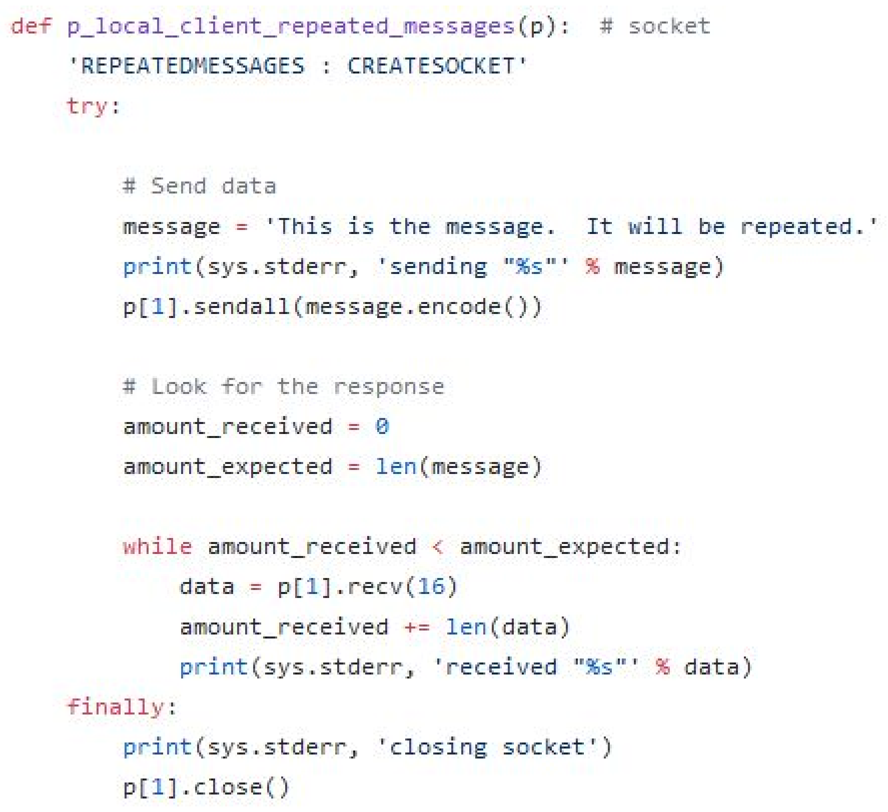
In general, this definition allows to input the parameters serveraddress host port and Socket in order to connect the local client`s socket to the port where the local server is listening.

This definition is equivalent to the following lines of code:

server_address = ('localhost', 10000)

print(sys.stderr, 'connecting to %s port %s' % server_address)

sock.connect(server_address)



```
def p_local_client_repeated_messages(p):  # socket
    'REPEATEDMESSAGES : CREATESOCKET'
    try:

        # Send data
        message = 'This is the message.  It will be repeated.'
        print(sys.stderr, 'sending "%s"' % message)
        p[1].sendall(message.encode())

        # Look for the response
        amount_received = 0
        amount_expected = len(message)

        while amount_received < amount_expected:
            data = p[1].recv(16)
            amount_received += len(data)
            print(sys.stderr, 'received "%s"' % data)
    finally:
        print(sys.stderr, 'closing socket')
        p[1].close()
```

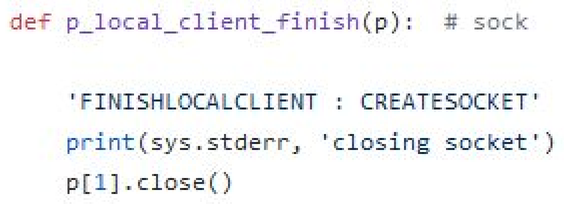
This code defines **REPEATEDMESSAGES** as a CREATESOCKET as it only takes a socket variable.

In general this definition allows a local client to sent repeated messages to the local server.

The only part where the socket is used is in the line:

p[1].close()   #which is equivalent to

sock.close()



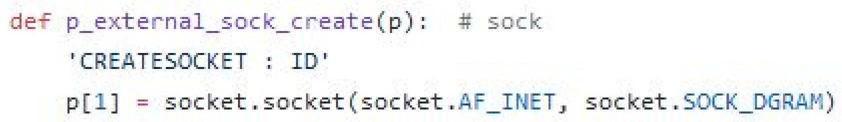
```
def p_local_client_finish(p):  # sock

    'FINISHLOCALCLIENT : CREATESOCKET'
    print(sys.stderr, 'closing socket')
    p[1].close()
```

This code defines **FI NISHLOCALCLIENT** as a CREATESOCKET as it only takes as socket variable .

In general this definition allows to terminate a connection for the local client.

This definition is equivalent to the following line of code:

socket.close()



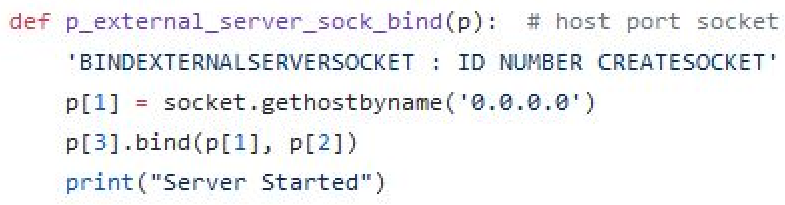
```
def p_external_sock_create(p):  # sock
    'CREATESOCKET : ID'
    p[1] = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

This code defines **CREATESOCKET** as an ID, which refer to a socket variable

In general this definition allows to create a socket for an external server.

This definition refers to the following line of code:

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)



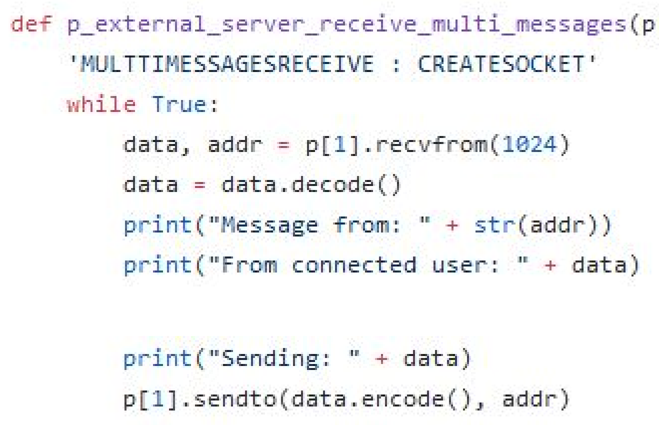
```
def p_external_server_sock_bind(p):  # host port socket
    'BINDEXTERNALSERVERSOCKET : ID NUMBER CREATESOCKET'
    p[1] = socket.gethostbyname('0.0.0.0')
    p[3].bind(p[1], p[2])
    print("Server Started")
```

This code defines **BINDEEXTERNALSERVERSOCKET** as an ID NUMBER CREATESOCKET, which refer to variables host Port socket

In general this definition binds the external socket to the port, assigns the host to the ip address of the local network.

This definition refers to the following lines:

host = socket.gethostbyname('0.0.0.0') s.bind((host, port))



```
def p_external_server_receive_multi_messages(p):
    'MULTTIMESSAGESRECEIVE : CREATESOCKET'
    while True:
        data, addr = p[1].recvfrom(1024)
        data = data.decode()
        print("Message from: " + str(addr))
        print("From connected user: " + data)

        print("Sending: " + data)
        p[1].sendto(data.encode(), addr)
```

This code defines **MULTIMESSAGESRECIEVE** as an CREATESOCKET which refers to a Socket variable .

In general this definition allows the server to receive multiple different messages from the external client.

The definition only uses the socket in only two lines:

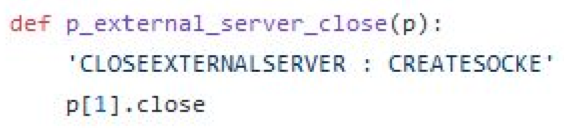
addr = p[1].recvfrom(1024)

p[1].sendto(data.endode(),addr)

Which refers to the lines:

addr = s.recvfrom(1024)

s.sendto(data.encode(), addr)



```
def p_external_server_close(p):
    'CLOSEEXTERNALSERVER : CREATESOCKE'
    p[1].close
```

This code defines **CLOSEEXTERNALSERVER** as an CREATESOCKET, which refers to a socket variable.

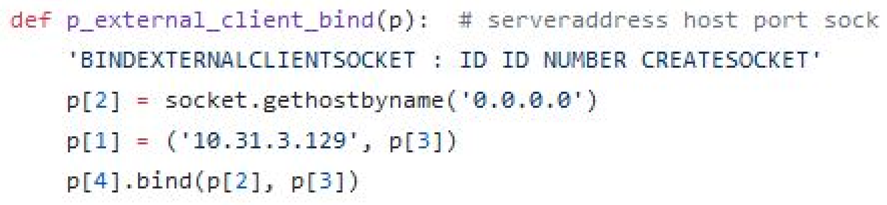
In general, this definition allows to terminate the external server process.

This definition only uses the line:

p[1].close()

Which refers to the line:

socket.close()



```
def p_external_client_bind(p):  # serveraddress host port sock
    'BINDEXTERNALCLIENTSOCKET : ID ID NUMBER CREATESOCKET'
    p[2] = socket.gethostbyname('0.0.0.0')
    p[1] = ('10.31.3.129', p[3])
    p[4].bind(p[2], p[3])
```

This code defines **BINDEXTERNALCLIENTSOCKET** as an ID ID NUMBER CREATSOCKET, which refer variables serveradress host Port Socket

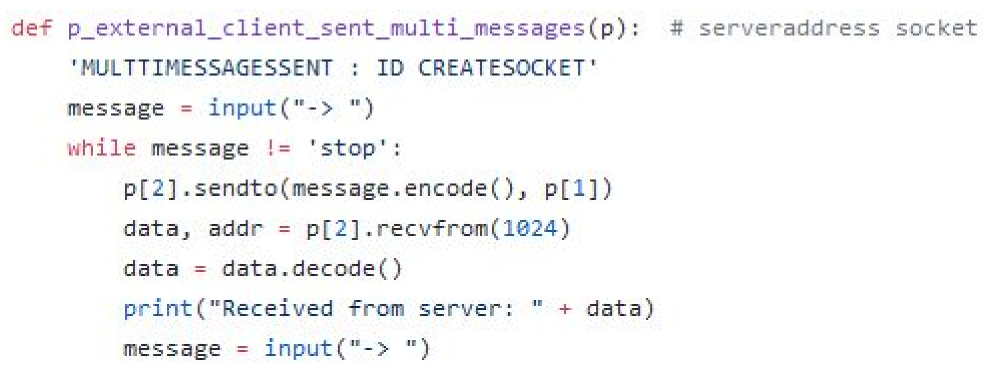
In general, this definition allows to input the parameters serveradress host port and Socket in order to connect the external client`s socket to an external server that is connected to RUMNET.

This definition refers to the following lines :

host = socket.gethostbyname('0.0.0.0')

    serveraddress = ('10.31.3.129', 4000)

socket.bind((host,port))



```python
def p_external_client_sent_multi_messages(p):  # serveraddress socket
    'MULTTIMESSAGESSENT : ID CREATESOCKET'
    message = input("-> ")
    while message != 'stop':
        p[2].sendto(message.encode(), p[1])
        data, addr = p[2].recvfrom(1024)
        data = data.decode()
        print("Received from server: " + data)
        message = input("-> ")
```

This code defines **MULTIMESSAGESENT** as an ID CREATESOCKET, which Id refers to the serveradress and CREATESOCKET refers to the external client socket.

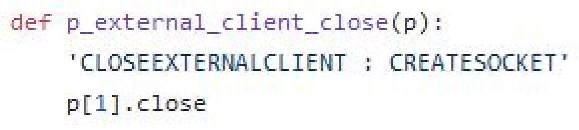
This definition allows the external “client device” to sent multiple messages to the external “server device”. It sends the first messages and and it enables the client to send multiple messages until he sends the message “stop”,

This definition only uses the socket in one line:

p[2].sento(message.encode(),p[1])

which refers to the line:

socket.sendto(message.encode(), serveraddress)



```python
def p_external_client_close(p):
    'CLOSEEXTERNALCLIENT : CREATESOCKET'
    p[1].close
```

This code defines **CLOSEEXTERNALCLIENT** as a **CREATESOCKET,** which refers to a socket variable.

This definition allows to terminate the connection of teh external client socket.

This definition only uses the socket in one line:

p[1].close()

Which refers to the line

socket.close()

WorkLoad:

Edgar Rivera:

- Worked on Documentation
- Worked on Parser
- Tested Servers

Edwin Borrero:

- Worked on Lexer
- Worked on Parser
- Tested Servers
- Uploaded Code to Git

Fabian Guzman:

- Video