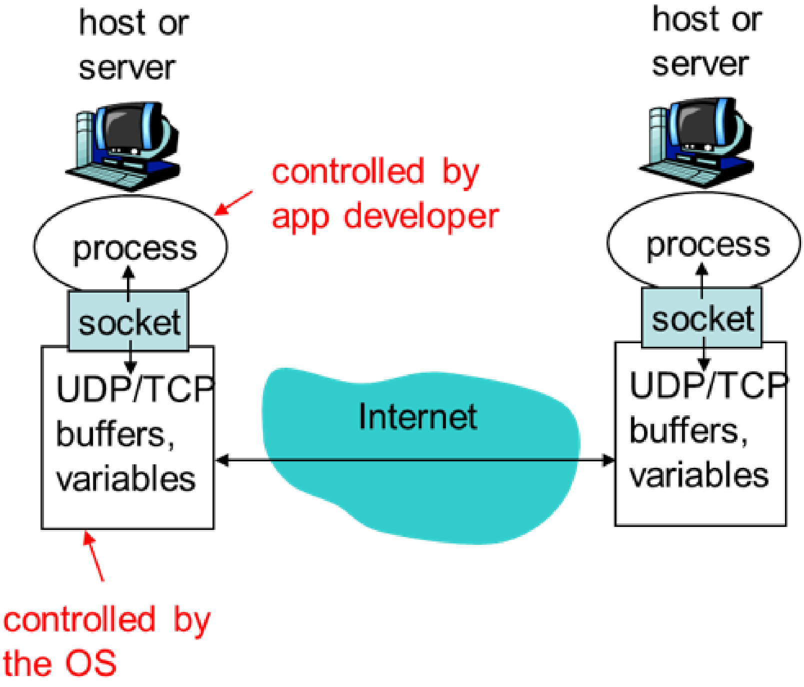
MIDDLE EAST TECHNICAL UNIVERSITY

METU EE444 Introduction to Computer Networks

HW1 – Socket Programming



You are going to submit your homework via **ODTUCLASS** as a **.zip** file containing the relevant source files (Client_process.py, Proxy_process.py and Server_process.py.) and your PDF report. Name your file as "HW1_studentid.zip".

Using code directly taken from any kind of resource, except those provided in ODTUCLASS, is prohibited. You can verbally discuss the homework with your friends but you should not exchange codes with each other or write your codes together. **Cheating and Plagiarism will result in zero grade, whereas disciplinary actions may also be taken. Late submissions are not allowed.**

1 How to Use Sockets

You can see the state diagram of TCP sockets in Figure 1.

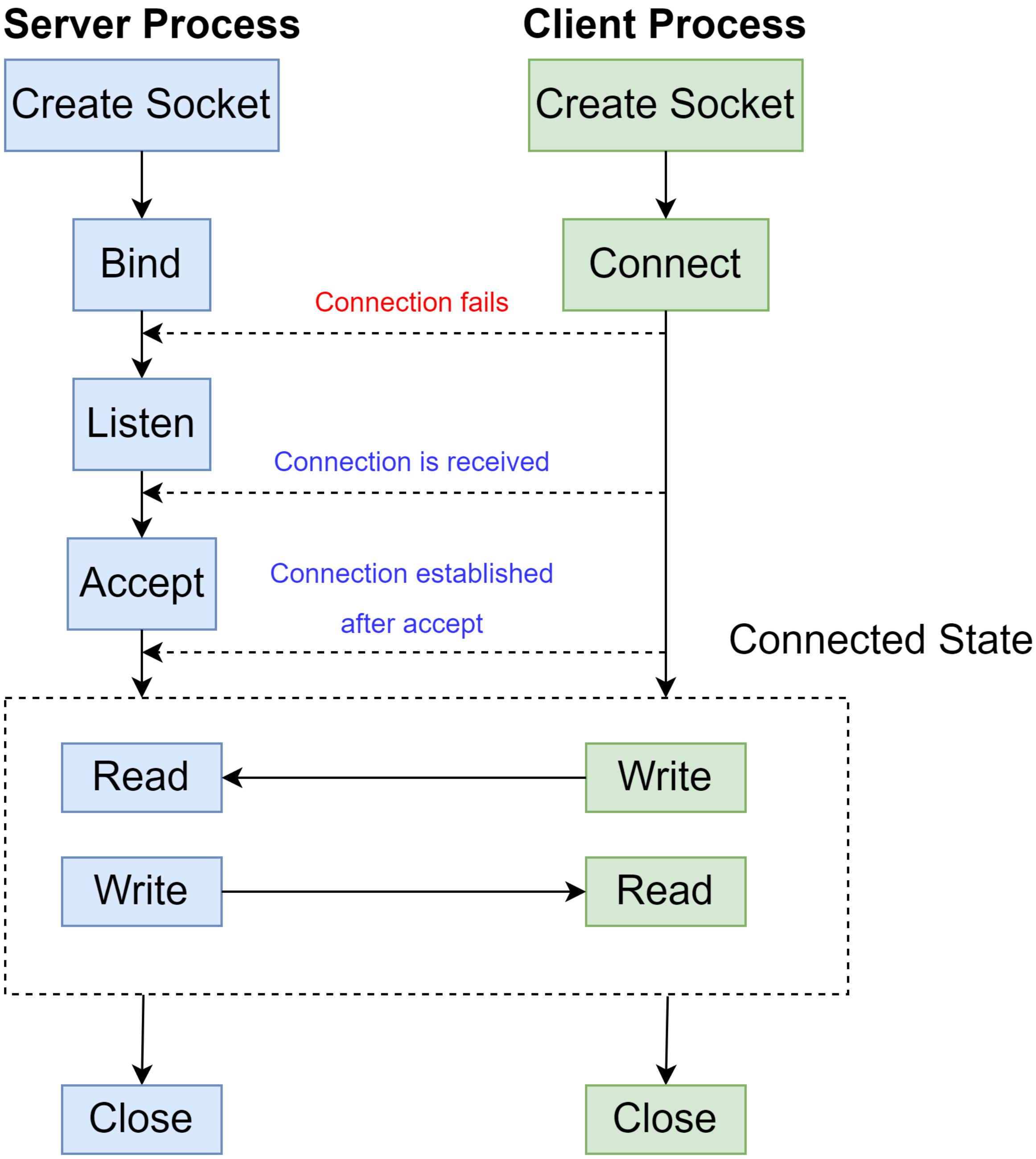


Figure 1: State diagram for TCP sockets

In this part, only the Python implementation of the sockets is covered but every programming language implements sockets in a very similar way. The socket library of python has many classes and functions which you can check here.

For this homework you only need to understand the basic functions of the socket library.

1. $\boxed{\text{socket.socket(family=AF\_INET, type=SOCK\_STREAM, proto=0, fileno=None)}}$
   Creates a socket object, default parameters are correct for this homework.

2. $\boxed{\text{socket.bind(address)}}$
   Binds the socket to an address which is (IP, PORT) pair.

3. $\boxed{\text{socket.close()}}$
   Closes the socket and all it's connections

4. $\boxed{\text{socket.listen([backlog])}}$
   Enable a server to accept connections. Listens for number of connections specified by backlog.

5. $\boxed{\text{socket.connect(address)}}$
   Connect to a remote socket at address, adress is (IP, PORT) pair.

6. $\boxed{\text{socket.accept()}}$
   Accepts and incoming connection. The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection.

7. $\boxed{\text{socket.recv(bufsize[, flags])}}$
   Receives specified number of bytes from the TCP buffer. Returns a byte object representing the received data.

8. $\boxed{\text{socket.send(bytes[, flags])}}$
   Sends specified number of bytes through the socket. Returns the number of bytes sent.

9. $\boxed{\text{socket.sendall(bytes[, flags])}}$
   Similar to send but this method continues to send data from bytes until either all data has been sent or an error occurs.

2 Simple Proxy Server (100%)

For this homework, you will implement a simple system consisting of 3 nodes. Namely, Client, Proxy and Server. Topology is very simple and can be seen in Figure 2.

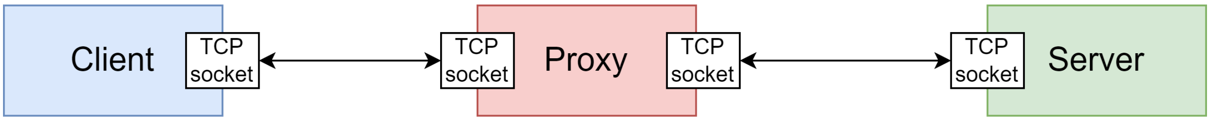


Figure 2: Network Topology

2.1 Operation Specifications

The Server will hold a list of 10 elements as described in Table 1. Each entry will consist of an index value ranging from 0 to 9 and data of a single integer. The Proxy should be consistent with the server, that is any update made to the Proxy's table should also be made in Server's table.

Table 1: Full table, stored in the Server

Index	**Data (Integer)**
0	Data0
1	Data1
2	Data2
...	...
9	Data9

The Proxy server will hold half of the table in its process as described in Table 2, think of it as a cached version of the Server's table. The Client will only communicate with the proxy server, if an element that is not present in the Proxy's table is required proxy server will communicate with the Server to get that element and add it to its table by overwriting the oldest table entry.

Table 2: Cached table, stored in the Proxy

Index	**Data (Integer)**
Index0	$Data_{Index0}$
Index1	$Data_{Index1}$
...	...
Index4	$Data_{Index4}$

Nodes will exchange proxy messages between them with the format as below.

```
OP=XXX;IND=Ind1,Ind2,..;DATA=Dat1,Dat2,...;
```

Notice that fields are separated with semicolons(;). OP field describes which operation to do on the table, for more detail check Table 3. IND field tells which of the indexes are required for the operation. The DATA field is for integer data either from the server for operations like "ADD" or as an update value from the client. Not all messages require every field. You can choose to omit the unused fields for certain operations. Details of the implementation are up to you

Response messages have the same form as request messages. For example response of the "ADD" message will have "ADD" as the operation code and contain the result in the "DATA" field.

Table 3: Operation Descriptions

OP Code	**Description**
GET	Request for the values of the specified indices Response should contain the values in the DATA field; Order of the values match order of the indices
PUT	Request for updating the values of the specified indices Request should contain the values in the DATA field Order of the values match the order of the indices
CLR	Clear the whole table for both the proxy and the back end server
ADD	Add the values of the specified indices Response should contained the added value in the data field For simplicity assume that maximum of 5 indices can be added

2.2 Implementation Specification

Client and the Proxy will be implemented in Python while the Server will be implemented in Matlab. Hence, you will be connecting 2 different programming environments through sockets.

The Client should have a terminal interface for creating every possible message. Specifics of the interface and how it takes the inputs from the terminal is up to you. Indicate any assumptions you made about the inputs.

Every node should print each message they send or receive through the sockets. Proxy and Server should print their tables whenever tables get changed.

2.3 Bonus (10%)

Define and Implement new operations that you think would be useful. You will get Bonus grade depending on your reasoning and implementation.

3 Deliverables

You will submit 3 code files. Namely, Client_process.py, Proxy_process.py and Server_process.m.

Additionally you will submit a PDF report detailing your implementation. Report does not need to long and it should not contain code segments unless it is relevant to the explanations.

4 Useful Know-Hows

In this section some libraries that will be helpful to you will be mentioned.

4.1 How to Parse Messages?

In Python there is ”re” library which has a split function as ”re.split()”. This function can split your message string into a list of strings separated by the specified separator characters.

For example: `dataSplit = re.split(’;|=’, InputStr)` will separate InputStr string into a list (array) of strings separated at (”;” and ”=”) characters.

In Matlab You can use the split function which works exactly the same as Python ”re.split()”

For example: `dataSplit = split(InputStr , [”=”,”;”]);` will separate InputStr string into a list (array) of strings separated at (”;” and ”=”) characters.

4.2 Sockets in Matlab

Matlab sockets are even easier than the Python sockets. Creating the server binds the socket and there is no explicit listening or accepting. Client gets created and connected in a single line of code. After the connection is made you can read and write using the relevant functions. You should only need 1 socket for the Matlab script in this homework.

Check the Matlab Socket Documentation, specifically tcpclient and tcpserver pages.