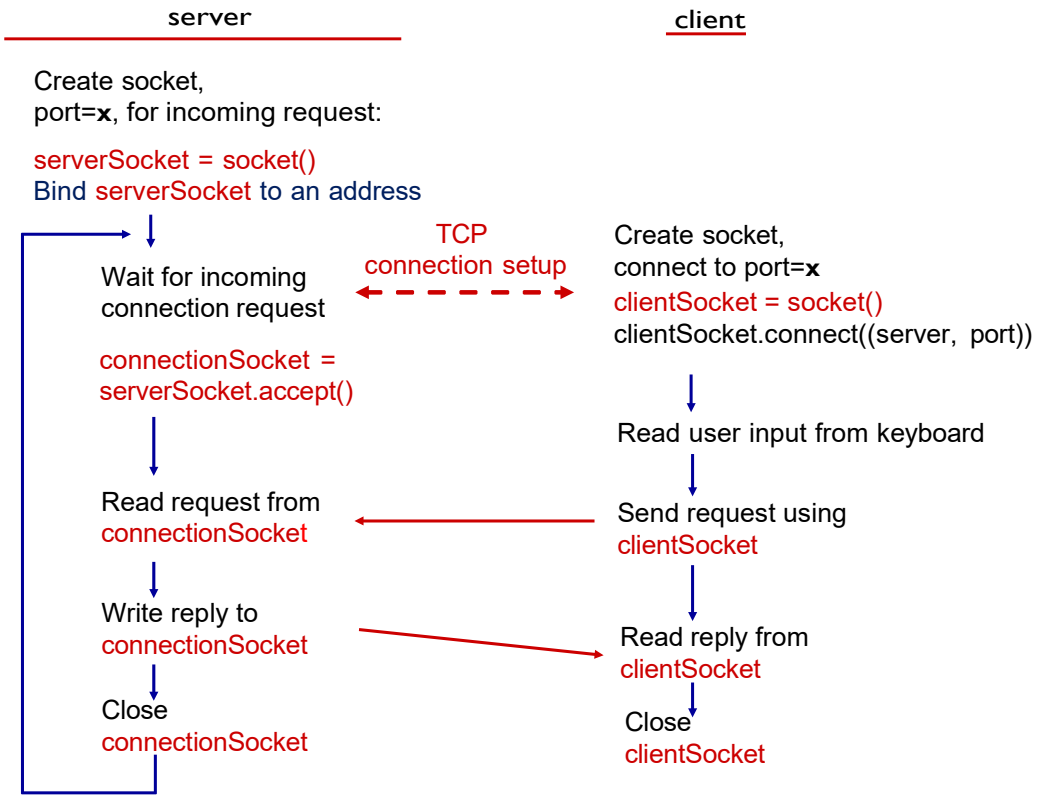
Durham University **Department of Computer Science**

Networks: Practical 2

The main aim of this practical is to help students to get familiar with socket programming. You should use Python 3.x for the implementation. You can do this practical with your own machine or the Linux machines from CIS (vega machines: https://durhamuniversity.sharepoint.com/Sites/MyDigitalDurham/SitePages/ServicePage.aspx?Service=%22Linux%20Timeshare%20(MIRA)%22).

Part 1 – Review Questions

1. List five non-proprietary Internet applications and the application-layer protocols that they use.
2. For a P2P file-sharing application, do you agree with the statement, "There is no notion of client and server sides of a communication session"? Why or why not?
3. What information is used by a process running on one host to identify a process running on another host?
4. Suppose you wanted to do a transaction from a remote client to a server as fast as possible. Would you use UDP or TCP? Why?
5. What is meant by a handshaking protocol?
6. For the client-server application over TCP shown below, why must the server program be executed before the client program? For a similar client- server application over UDP, why may the client program be executed before the server program?



Part 2 – Python Socket Programming

For this task, please use Python 3.x. You may wish to refer to the example client and server programs provided in the Duo materials for Lecture 3.

Other useful materials to help you complete this task:

- John Goerzen, Foundations of Python Network Programming, APRESS, 3rd edition
- https://docs.python.org/3/howto/sockets.html
- https://pymotw.com/3/socket/tcp.html

Task 1

1. Use the socket API to develop a pair of client and server programs, where the client can send many simple messages one by one to the server. The client can stop running at any time when it does not want to send any further messages to the server.
2. Whenever the server received a message from the client, the server should print out the message on its console screen.
3. Ask a student which socket he/she is using for his/her server program, modify your own client program to connect to this socket. See whether your client program can successfully communicate with the server program of that student and observe whether proper results can be obtained.[1]

Task 2

1. Network security is important in general. It is not good to allow your server program to be used by any other person without any authorization. Modify your server program (and client program if you think it is applicable) to avoid any student being able to communicate with your server program using his/her client program even if that student knows which socket that you are using.
2. Ask a student to verify whether your modification works.

Task 3

1. Set up a simple table at the server to hold records of {client ID, number of messages} information. You may consider using a 2D array to set up the table. The table tells the maximum number of messages each client can send to the server.
2. Modify the client program to send a client ID to the server for identifying itself prior to any communication.
3. When a client sends a message to the server, the server should check against the table to see whether this client has already sent enough number of messages. If yes, the server should tell the client not to send any more messages to the server.
4. You may use the client program to send out even more messages to test whether the server program responses properly.
5. Run the client program by using different client IDs (or even a client ID that cannot be found in the table at the server.) and again test whether the server program responses properly.

Optional tasks

Familiarize yourself with the UNIX programming environment using mira1.dur.ac.uk or mira2.dur.ac.uk.

UNIX Tutorial for Beginners: http://www.ee.surrey.ac.uk/Teaching/Unix/

Login mira1.dur.ac.uk using putty (Windows) or ssh (Mac/Unix)

(In case you need to download putty:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)

Useful commands:

1. Check out which network ports are being used: netstat –a | more
2. Run a program in background mode: <program name> &
3. Check out your own running processes (with process ID): ps
4. Check out all running processes: ps –df
5. Check out specific running processes: ps –df | grep <keyword>
6. Kill process: kill -9 <process ID>
7. Show current path: pwd