CSC3002 – Networks Assignment 1 – 2023
Socket programming project

Department of Computer Science
University of Cape Town, South Africa

February 19, 2023

This assignment is on networked applications where you will learn the basics of protocol design and socket programming for TCP connections in Python: how to create a socket, bind it to a specific address and port, as well as send and receive messages/files. You will develop a client-server application in groups of three students. This document describes the context/requirements (Section 1), what to submit (Section 3), and some basic information about socket programming (Section A).

1. Application Description

There are a number of file sharing protocols, many of them employing what is commonly referred to as P2P or Peer-to-Peer sharing. Usually, sharing files is done directly between two nodes in a network. A different approach for sharing is to use the client-server model where clients upload and download files from a shared server.

In this assignment, you are to design and implement a client-server file sharing application that makes use of TCP sockets. The server needs to be able to receive and send files to clients. Client should be able to upload files to the server, and clients should be able to indicate if the file being uploaded is 'open' or 'protected'; where an 'open' file is visible and downloadable by any client, and a 'protected' file is only visible and downloadable by clients that have the right 'key' for the file. Clients should be able to query the server for listing of available files and should be able request (download) files from the server.

The system should include a file validation mechanism, where the sender includes extra validation information within each message, and where the receiver is able to check that a file is exactly as it was sent by the source (i.e., file has not been altered in transit).

Your web server should accept and parse a client request, get the requested file from the server's file system, create a response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present, or the client is not permitted to access the requested file (i.e., right key

not provided by client), then the server should send back an appropriate error message back to the client.

You should implement your client to use a TCP connection to interact with server and and display the server responses. The client implementation should allow for command line arguments, such as specifying the server IP address, the port at which the server is listening, and the request, e.g requesting to download a particular file stored at the server.

2. Protocol Design and Specification

An important step in protocol design involves defining the framework of communication for the intended application. This entails specifying requirements and constraints such as: whether real-time interaction is expected; reliability (ie if we need to verify/check that every message is delivered correctly); and authentication/confidentiality issues, among others. In this assignment, you are to design the application that takes into consideration privacy/confidentiality, by allowing users indicate file sharing permissions, e.g should a file be visible/downloadable by anyone querying? Or, should a file be visible/downloadable (or not) to certain specific clients? Or can a shared secret-key be used to access/download files? The application protocol should be designed to include some of these features.

Specification of protocol messages involves defining the types and structure of messages. Three types of messages can be defined; commands, data transfer, and control. Command messages define the different stages of communication between parties, such as the initiation or termination of communication. Data transfer messages are used to carry the data that is exchanged between parties, and such data could be fragmented into several messages. Control messages manage the dialogue between parties, including such aspects as message acknowledgements, and retransmission requests.

The message structure constitutes at least the header and body. The header, whose structure must be known to both sides of a communication link, may contain fields that describe the actual data in the message. Some of the fields/information contained in the header might be the message type, the command, recipient information, and sequence information. The header generally has fixed size and contains clues that should help the receiver to understand the rest of the message.

The last aspect of the protocol design will be the communication rules that specify the sequence of messages at every stage of communication. This requires clearly specifying messages and reactions for every communication scenario. You will need to represent such rules with sequence diagrams (at lease two sequence diagrams will be required, one for upload process, and another for download process).

3. What you need to submit

As a group, design/specify a protocol to support the application, and implement the server. As individuals within each group, implement a client that can interact with the group's server. You will be required to submit the following:

1. The group server code, and individual client code, with proper inline documentation (comments)
2. A report (max 6 pages) on the design and functionality of your chat application (individual sections on the client implementation).
3. In addition, the report needs to include:
   a) A list of features with a brief explanation for their inclusion
   b) A protocol specification, detailing the message formats and structure. You are required to include sequence diagram(s).
   c) Screenshots of the application revealing its features.
4. Oral presentation to be scheduled with the TAs and Tutors (oral to be done on day after submission deadline)

A. Multi-threaded Client/Server Applications—Sockets Programming

A.1. What is a socket?

A socket is the one end-point of a two-way communication link between two programs running over the network. Running over the network means that the programs run on different computers, usually referred as the local and the remote computers. However one can run the two programs on the same computer. Such communicating programs constitutes a client/server application. The server implements a dedicated logic, called **service**. The clients connect to the server to get served, for example, to obtain some data or to ask for the computation of some data. Different client/server applications implement different kind of services.

To distinguish different services, a numbering convention was proposed. This convention uses integer numbers, called port numbers, to denote the services. A server implementing a service assigns a specific port number to the entry point of the service. There are no specific physical entry points for the services in a computer. The port numbers for services are stored in configuration files and are used by the computer software to create network connections.

A socked is a complex data structure that contains an internet address and a port number. A socket, however, is referenced by its descriptor, like a file which is referenced by a file descriptor. That is why, the sockets are accessed via an application programming interface (API) similar to the file input/output API. This makes the programming of network applications very simple. The two-way communication link between the two programs running on different computers is done by reading from and writing to the sockets created on these computers. The data read from a socked is the data wrote into the other socket of the link. And reciprocally, the the data wrote into a socket in the data read from the other socket of the link. These two sockets are created and linked during the connection creation phase. The link between two sockets is like a pipe that is implemented using a stack of protocols. This linking of the sockets involves that internally a socket has a much more complex data structure, or more precisely, a collaboration of data structures. Thus, a socket data structure is more than just an internet address and a port number. You have to imagine a socket as a data structure that contains at least the internet address and the port number on the local computer, and the internet address and the port number on the remote computer.

A.2. How is a network connection created?

A network connection is initiated by a client program when it creates a socket for the communication with the server. To create the socket, the client calls the Socket constructor and passes the server address and the the specific server port number to it. At this stage the server must be started on the machine having the specified address and listening for connections on its specific port number.

The server uses a specific port dedicated only to listening for connection requests from

clients. It can not use this specific port for data communication with the clients because the server must be able to accept the client connection at any instant. So, its specific port is dedicated only to listening for new connection requests. The server side socket associated with specific port is called server socket. When a connection request arrives on this socket from the client side, the client and the server establish a connection. This connection is established as follows:

1. When the server receives a connection request on its specific server port, it creates a new socket for it and binds a port number to it.

2. It sends the new port number to the client to inform it that the connection is established.

3. The server goes on now by listening on two ports:
   - it waits for new incoming connection requests on its specific port, and
   - it reads and writes messages on established connection (on new port) with the accepted client.

The server communicates with the client by reading from and writing to the new port. If other connection requests arrive, the server accepts them in the similar way creating a new port for each new connection. Thus, at any instant, the server must be able to communicate simultaneously with many clients and to wait on the same time for incoming requests on its specific server port. The communication with each client is done via the sockets created for each communication.

TCP is a connection-oriented protocol. In order to communicate over the TCP protocol, a connection must first be established between two sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once the two sockets are connected, they can be used to transmit and/or to receive data. When we say "two sockets are connected" we mean the fact that the server accepted a connection. As it was explained above the server creates a new local socket for the new connection. The process of the new local socket creation, however, is transparent for the client.

The datagram communication protocol, known as UDP (user datagram protocol), is a connectionless protocol. No connection is established before sending the data. The data are sent in a packet called datagram. The datagram is sent like a request for establishing a connection. However, the datagram contains not only the addresses, it contains the user data also. Once it arrives to the destination the user data are read by the remote application and no connection is established. This protocol requires that each time a datagram is sent, the local socket and the remote socket addresses must also be sent in the datagram. These addresses are sent in each datagram.