Assignment 1 (100 points)

Anirudh Sivaraman

2021/09/17

General instructions

This assignment is done in two parts: the multiple choice section and the code section. Both are submitted on Gradescope.

1 (60 points) Multiple Choice

See Gradescope for questions.

2 (40 points) Coding Assignment

(1 points) Turn in all code that will be edited by you. This includes:
concatenation_server.py
concatenation_client.py
rpc_server.py
rpc_client.py
relay.py
sender.py
receiver.py

2.1 (13 points) TCP string concatenation server

To start off with socket programming, we'll build a really simple networked application using sockets. Here, a client sends some data to the server, which concatenates another piece of data to that data and sends it back to the client. We have provided starter code for the client and server as a starting point for this assignment. Submit the final versions of your client and server programs.

2.2 (13 points) UDP RPC server

We'll now build a simple remote procedure call (RPC) system that allows a client to invoke methods remotely on a server machine. We'll support one RPC method where the client sends "prime(N)" and the server sends back yes or no depending on whether N is prime or not. You should use UDP to communicate between the client and the server. Use the provided starter code as a starting point, and submit the final versions of both your client and server programs.[1]

[1] UDP is unreliable. So, an application that calls recv() on a UDP socket may deadlock waiting for lost data that never arrives. Any real UDP application needs a timeout so that it doesn't wait indefinitely. This timeout can be implemented using select (see lecture 3 notes). We will ignore this complication here because losses are rare when communicating between two sockets on a single machine.

2.3 (13 points) TCP relay

We'll build a simple TCP relay in this exercise. A relay is a program that relays data between two other programs, often inspecting the data it is relaying for security and accounting purposes. A TCP relay works as follows. Let's assume there are two TCP end points (*A* and *B*), connected as follows:

```
A <---> B
```

The double-headed arrow indicates that communication can happen in both directions between A and B: A can send to B and B can send to A.

Now, if we introduce a relay *R* between *A* and *B*, this picture is modified to:

```
A <----> R <----> B
```

Now *R* takes bytes coming in from *A* and sends them to *B* and vice versa. This arrangement is at the heart of proxying, something you may have heard of. In such an arrangement, *R* would be called a proxy server.[2] Proxy servers are used for several reasons. For instance, *R* could require *A* to authenticate with *R* before communicating with *B*. *R* could be used to account for how much traffic *A* sends out of the network. It could also be used to inspect the traffic between *A* and *B* to ensure it doesn't contain anything malicious.

How do we build this relay? For this assignment, relay *R*'s work is to detect any bad words in the strings being transmitted and replace them with corresponsing good words. The bad words are: $['virus', 'worm', 'malware']$ and their corresponding good words are: $['groot', 'hulk', 'ironman']$. Note that the length of the bad word and its corresponding good word is the same. This means after the relay replaces the bad words with the good words, the length of the string being transmitted does not change.

For this exercise, you'll be required to turn in three programs, one each for *A*, *B*, and *R*. We'll also simplify the requirement to one directional communication from A to R to B, and not worry about communication from B to R to A. But, you cannot connect *A* and *B* directly; you have to connect them through the relay *R*.

We have provided starter code for each of the three components here: the sender (A), the receiver (B), and the relay (R). Along with the starter code we also provide you with a test file with 200 randomly generated strings that you can use to test your programs. These test strings can have zero or more bad words in them and their length is 200.

Please call the relay, sender, and receiver (in that order) as given below.

```
$: python3 relay.py <relay_port>
$: python3 sender.py <relay_port> <test_filename>
$: python3 receiver.py <relay_port>
```

Please note that the sender only reads test cases from a file. In case you want to have custom test cases you can create a new test file with the custom strings in them. Each line in the test file makes one test string.

```
Example:
Input string: gXPnDworm9s86Fq80v26TeCpCAvirusqsMU6m8Mt0SD
Relay receives the string and updates it to: gXPnDhulk9s86Fq80v26TeCpCAgrootqsMU6m8Mt0SD
Receiver should get: gXPnDhulk9s86Fq80v26TeCpCAgrootqsMU6m8Mt0SD
```

Submit the final version of each of the three programs.

[2] In a real proxy server, a single *R* might handle multiple *A*s and *B*s on each side, but we'll handle only a single *A* and *B* in our exercise.