CMPU4021 Distributed Systems – Labs

Week 2 – Socket Communication in Python

Learning Outcomes:

- Be able to open TCP and UDP sockets with Python.
- Be able to implement a simple distributed system: i.e. client server applications in Python.

Tasks

Python Socket Library

For socket programming in Python, built-in Python `socket` library provides functions, constants, and classes that are used to create, manage and work with sockets. Some of the important server socket methods are:

- `socket()`: Creates a new socket.

- `listen()`: is used to establish and start TCP listener.
- `bind()`: is used to bind-address (host-name, port number) to the socket.
- `accept()`: is used to TCP client connection until the connection arrives.
- `connect()`: is used to initiate TCP server connection.
- `send()`: is used to send TCP messages.
- `recv()`: is used to receive TCP messages.
- `sendto()`: is used to send UDP messages
- `close()`: is used to close a socket.

Tasks:

T1: Writing a simple TCP echo client/server application

Examine `PT1\1_13a_echo_server.py` and `PT1\1_13b_echo_client.py` files. In this example, a server will echo whatever it receives from the client. Python `argparse` is module is used to specify the TCP port from a command line. Both the server and client script will take this argument.
We first create the server: start by creating a TCP socket object. The socket is bound to the given port on the local machine. In the listening stage, we listen to multiple clients in a queue using the backlog argument to the `listen()` method. The server waits

for the client to be connected and send some data to the server. When the data is received, the server echoes back the data to the client.

On the client side code, a client socket is created using the port argument and connect to the server. Then, the client sends the message, *"Test message. This will be echoed"* to the server, and the client immediately receives the message back in a few segments. Two try-except blocks are constructed to catch any exception during this interactive session.

To run these examples launch the server script in one terminal:

```
> python 1_13a_echo_server.py --port=9900
Starting up echo server on localhost port 9900
Waiting to receive message from client
```

Run the client from another terminal:

```
>python 1_13b_echo_client.py --port=9900
Connecting to localhost port 9900
Sending Test message. This will be echoed
Received: Test message. Th
Received: is will be echoe
Received: d
Closing connection to the server
```

Upon receiving the message from the client, the server will also print a message similar to the following:

```
Data: Test message. This will be echoed
sent Test message. This will be echoed
bytes back to ('127.0.0.1', 42961)
Waiting to receive message from client
```

T2.Writing a simple UDP echo client/server application

Examine `PT2\1_14a_echo_server_udp.py` and `PT2\1_14b_echo_client_udp.py` files.

This example is similar to the Task 1, except this one is with UDP. The method `recvfrom()` reads the messages from the socket and returns the data and the client address.
The client side code, a client socket is created using the port argument and connect to the server, as we did for the TCP example. Then, the client sends the message, "`Test message. This will be echoed`". The client then receives the message back in a few segments.

To run these examples launch the server script in one terminal:

```
>python 1_14a_echo_server_udp.py --port=9900
Starting up echo server on localhost port 9900
Waiting to receive message from client
```

Run the client from another terminal as follows:

```
>python 1_14b_echo_client_udp.py --port=9900
Connecting to localhost port 9900
Sending Test message. This will be echoed
received Test message. This will be echoed
Closing connection to the server
```

Upon receiving the message from the client, the server will also print something similar to the following message:

```
received 33 bytes from ('127.0.0.1', 43542)
Data: Test message. This will be echoed
sent 33 bytes back to ('127.0.0.1', 43542)
Waiting to receive message from client
```

T3. Create a Python client-server application using UDP. The aim of the application is to guess a number. When the server is started, it stores a random number between 1 and 100...

```
import random
#returns a number between 1 (included) and 100 (not included)
print(random.randrange(1, 100))
```

The client reads numbers in from the user and sends these to the server. The server responds with a string saying HIGHER, LOWER to CORRECT.
The client can continue entering values until it gets CORRECT returned.

If you do not finish this task in the lab – you can continue it next week or finish at home.

References

- Chapter 1, Kathiravelu P., Sarker F., Python Network Programming Cookbook, Second Edition, Packt Publishing Ltd, 2017
- https://www.python.org/
- https://docs.python.org/3/library/ipc.html
- https://www.w3schools.in/python/network-programming