CPNT-217

Introduction to Network Systems

Socket Programming

Weight: 6% **Marks: /10**

Student Name:

Student ID:

Date:

This page was intentionally left blank.

Table of Contents

Lab Title 1

Introduction 1

Equipment and Materials 1

Procedure 1

Marking Criteria 1

References 2

Lab Title

Introduction

In this lab exercise, learners will explore the creation and usage of sockets in Python programming. Sockets are in integral part of many applications, allowing the application to send and receive data through specified ports both with and without action from the user. A socket combines a triad of protocol, IP address and port number to create a software construct that the application will use for network access.

Note: This lab was developed using Python 3.9.13. Please ensure that your system is using a version of Python which is compatible.

Equipment and Materials

- 1 desktop or laptop computer with network access
- 1 Python programming environment (Students may use an environment of their choice, suggestions of environment include Notepad++, PyCharm, Microsoft Visual Studio, or the Python IDE) Please be sure to include the Python installation folders in the PATH of your Windows environment. Procedure

Submitting your work

Create a separate document for your lab submission. Use the new document to record your responses and notes, as well as any required screen captures. (Please label your screen captures.)

Submit your lab document on D2L including responses to all questions. Be prepared to discuss the reflection questions in class.

Procedure:

Part 1: Basic socket creation

In this section, we will create a small Python server in order to demonstrate the basic functions of creating and opening a socket to communicate on a network.

1. In your chosen development environment, open a new Python file. Follow the steps listed to create your server application.
2. As in most software development projects utilizing Python, we will be using an imported library or module to take advantage of functions and methods that have been previously

created inside our project. In this case, we will import the socket library. This library will provide us with access to various methods that relate to how sockets work.

```
import socket #imports the 'socket' library to use in our code
```

3. Next, we are going to create a variable called port. This variable will hold the port number we have chosen for our server application to use.

```
port=35685 #creating the 'port' variable
```

4. We have a number of options that we can choose from to create a socket with our server, such as whether to use IPv4 or IPv6, and whether we want to use a connection-oriented Transport protocol (TCP) or connectionless (UDP). In this case, we will use IPv4 (indicated by the AF_INET object, and TCP (indicated by the SOCK_STREAM object).

```
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM) #sets up the socket to use IPv4 (AF_INET) and TCP (SOCK_STREAM)
```

5. Now that we have set our application to use IPv4 and TCP for creating our socket, it is time to finish the creation. To do this, we are going to bind the IP address of our system to the port number we have assigned.

```
addr=(socket.gethostbyname('localhost')) #identifies the IP address using the hostname
```

```
s.bind((addr,port)) #creates the socket by binding the port to the IP address
```

(The first line finds the IPv4 address of the local system, while the second line binds the port to the IP address to create the socket.)

6. After we create the socket, we need to tell the application what it should do with the socket. In this case, we want it to listen to the socket and wait for a connection.

```
s.listen(5) #instructs the socket to listen for a connection from up to 5 clients at one time
```

We also want to make sure that the server is listening, so we will have the server print a message. This is a reminder to the user of what the server is doing.

```
print("[+] Server listening...") #indicates that the server is listening
```

7. Since the server will now be listening for connections to the socket we have created, we need to give the server instructions for what to do when it detects a connection. We will create a while loop with the execution condition of having the socket be TRUE. (Meaning the socket has a connection attempt incoming.)

```
while True: #gives instructions on what to do if there is a connection to the socket
```

8. The first thing we want the server application to do when there is a connection attempt is to accept the connection.

```
    conn,addr=s.accept() #accepts the connection from the remote client
```

9. We want our application to record the address and port that the client is using to connect, so we will print this information to the console. (We could also save this information to a file for logging purposes.)

```
    print(f"connection to {addr[0]} on {addr[1]}") #displays where the connection is originating, and on what port
```

10. Lastly, we want to send a message to the client, to indicate that the connection was successful. We will encode this message using the utf-8 format.

```
    conn.send(bytes("Socket Programming in Python", "utf-8 ")) # sends a message for the client to display to the user
```

Now our server application is complete. We have created a socket and instructed the application what actions to take if a connection occurs.

```
import socket #imports the 'socket' library to use in our code
port=35685 #creating the 'port' variable
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM) #sets up the socket to use IPv4 (AF_INET) and TCP (SOCK_STREAM)
addr=(socket.gethostbyname('localhost')) #identifies the IP address using the hostname
s.bind((addr,port)) #creates the socket by binding the port to the IP address
s.listen(5) #instructs the socket to listen for a connection from up to 5 clients at one time
print("[+] Server listening...") #indicates that the server is listening

while True: #gives instructions on what to do if there is a connection to the socket
    conn,addr=s.accept() #accepts the connection from the remote client
    print(f"connection to {addr[0]} on {addr[1]}") #displays where the connection is originating, and on what port
    conn.send(bytes("Socket Programming in Python", "utf-8 ")) # sends a message for the client to display to the user
```

Note: You may want to use a search engine to help you with the answers to these questions.

What change would you make if you wanted the application to use IPv6 instead of IPv4?

What are the three options available when we are selecting the protocol for our socket to use?

Part 2: Connecting to a Socket

In part 2, we will create a client application to connect to our server and receive a message.

1. Create a new Python file in your development environment. Follow the steps to code your client application.
2. The beginning of our client application will be the same as our server because we need to use a socket on this side of the network connection as well. We will use the same port number because we want our client to send to the server on the port it is listening on.

```
import socket #imports the 'socket' library to use in our code
```

```
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM) #sets up the socket to use IPv4 (AF_INET) and TCP (SOCK_STREAM)
```

```
port=35685 #creates the port variable
```

3. In our lab, we will be running the client and the server from the same machine, so we will set up our client to use the s.connect method to connect and send a message to the localhost address, to the port we specified in the server application.

```
s.connect(('localhost', port)) #sets up the connection using a socket
```

4. Next, we will set up a variable to contain any message that is sent by the server application. We will call it msg.

```
msg=s.recv(1024) #creates a variable to contain any message sent by the server
```

5. We will want to display our message once it has been received, so we will print it to the console, decoding the utf-8 formatting that was applied by the server.

```
print(msg.decode("utf-8")) #prints the message received from the server
```

We have now created our client application. When executed it will create a socket to connect to the server, receive any message that is sent and display it on the console. After it completes that process, it will end.

What is the maximum size of message that our client can receive?

How could we change our client to allow it to connect to a server on a different host on the network?

Part 3: Testing the connection

With the server and client applications created, they can now be used to illustrate the connection between them. We will do this by running the server and the client separately. (These instructions assume you have named your applications server.py and client.py. If you have used alternate names, substitute those names in these commands.)

1. Open the Start Menu and type *powershell*. This will open the Windows Powershell window, which is a command line tool that Microsoft includes in Windows operating systems.
2. In Powershell, navigate to the folder containing your Python applications, and execute the server using the command *python server.py.*
3. The server should begin listening and should print the message in the powershell window that it is listening. If you receive an error, pay attention to the listed error and the line number, close Powershell and double-check your development environment to locate the error and correct it.

   Note: Do not proceed until the server is working correctly.
4. Open the Start Menu and type *cmd*. This will open a Windows command prompt similar to Powershell, but separate.

5. In the command prompt window, navigate to the folder where your client application is saved and use the command *python client.py* to execute the client.
6. You should receive feedback in both the Powershell window and the command prompt window indicating that the connection was made, including information on the server indicating where the connection originated from.

You should now have an understanding of what sockets are and how they are created.

What is the purpose of creating sockets inside applications?

Reflection: How many sockets do you think you use each day?

Challenge Mode: (Optional content)

Work with a classmate to allow your client application to connect to a server on a separate machine instead of the original host! Try it with a hard-coded IPv4 address, or with user input!

Marking Criteria

1 point per correct answer – 5 points

1 point per answer for using developed sentences – 5 points

References

<Reference (APA format)

