COMP3234 Computer and Communication Networks / ELEC3443 Computer Networks

Lab 1: Socket Programming

Introduction

In this lab, we will practice Python socket programming. We will first try out the example client and server programs in the lecture, and then implement a simple file transfer application.

Python Socket Programming

Follow the instructions on https://www.python.org/downloads/release/python-380/ to install the latest Python release on your operating system.

To use sockets in your Python programs, you must include the Socket Module in your programs, which contains Low-level networking interface (close to the BSD API):

```
from socket import *
```

or

```
import socket
```

(check out their differences on page 11 of lecture notes on Socket Programming)

The following is a list of often used Python socket APIs for your reference (see more at https://docs.python.org/3/library/socket.html):

s = socket.socket (socket_family, socket_type, protocol)	create a socket: • socket_family : AF_INET (IPV4) or AF_INET6 (IPV6) (commonly used two; see others at https://docs.python.org/3/library/socket.html) • socket_type: SOCK_STREAM (TCP), SOCK_DGRAM (UDP) (commonly used two; more at https://docs.python.org/3/library/socket.html) • protocol: This is defaulting to 0

Server socket functions:

s.bind(address)	bind address to socket: • address is (hostname or IP address, port number) for IPV4 (AF_INET); • address is (host, port, flowinfo, scopeid) for IPV6 (AF_INET6)
s.listen([backlog])	used by TCP server program; set up and start TCP connection listener: • backlog specifies the max. no. of

	incoming connection requests that can be queued while waiting for server to accept them, which is optional
s.accept()	used by TCP server program to accept TCP client connection: • return value: a pair, (conn, address), where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection

Client socket functions:

s. connect(address)	establish a connection to a server socket at address, which is waiting at accept()

Socket functions that both client and server can use:

s.recv(bufsize[, flags])	receive TCP stream data from the socket: • bufsize: maximum amount of data to be received at once • flags: optional; see more at https://manpages.debian.org/buster/manpages-dev/recv.2.en.html • return value: a bytes object representing the data received
s.send(bytes[, flags])	send TCP stream data to the socket: • bytes: the data object to send • flags: optional, same meaning as for recv() above • return value: the number of bytes sent
s.recvfrom(bufsize[, flags])	receive UDP datagrams from the socket: : • bufsize: maximum amount of data to be received at once • flags: optional, same meaning as for recv() above • return value: a pair, (bytes, address), where bytes is an object representing the data received and address is the address of the socket sending the data
s.sendto(bytes[, flags], address)	send UDP datagrams to the socket: • bytes: the data object to send • flags: optional, same meaning as for recv() above • address: address of the destination socket • return value: the number of bytes sent
s.close()	close the socket
socket.gethostname()	return a string containing the hostname of the machine where the current program is executing.
s.getsockname()	return the current socket's own address: • return value: address in a format according to the socket's address family

Run example programs

We have provided source code of a few examples for socket programming in socketprog_examples.zip. Please try them out as follows.

Example 1 (TCPSocket-1): This is the TCP server (sequential) and client example in the lecture slides.

- **Launch one terminal** and switch to the directory of TCPSocket-1. Run the server program as follows:

```
python3 TCPServer.py
```

You will see the prompt in the terminal "The server is ready to receive", which means the server is listening.

- **Launch another terminal** and switch to the directory of TCPSocket-1. Run the client program as follows:

```
python3 TCPClient.py
```

You will see the prompt "Input a lowercase sentence:". You can enter such a sentence and press "enter". Then you will see an uppercase sentence returned from the server.

Example 2 (TCPSocket-2): This is equivalent implementation of the TCP server (sequential) and client example in the lecture slides, where instead of using "from socket import *" to import APIs in the socket module, we use "import socket". Compare the difference between the programs in Example 1 and Example 2.

Run the example following the same steps as given in Example 1. (You can use control+C to kill a running server.)

Example 3 (TCPSocket-3): This is another implementation of the TCP server (sequential) and client example in the lecture slides, where we add a number of exception handling codes. Read more about Python error and exception handling at https://docs.python.org/3/tutorial/errors.html, exception socket.error at https://docs.python.org/3/library/socket.html#socket.error, Python sys module at https://docs.python.org/3/library/sys.html .

Run the example following the steps given in Example 1. When you see the prompt "Input a lowercase sentence:", try using "control+c" instead of entering a sentence; then you will see error prompt on both terminals running client and server programs.

Example 4 (TCPSocket-4): This is the TCP server (concurrent) and client example in the lecture slides, where the server can handle concurrent connections through threading.

- **Launch one terminal** and switch to the directory of TCPSocket-4. Run the server program as follows:

```
python3 TCPServer.py
```

You will see the prompt in the terminal "The server is ready to receive", which means the server is listening.

- **Launch the second terminal** and switch to the directory of TCPSocket-4. Run the client program as follows:

```
python3 TCPClient.py
```

- **Launch the third terminal** and switch to the directory of TCPSocket-4. Run the client program as follows:

```
python3 TCPClient.py
```

Now two connections are set up with the same server. You can enter sentences on the second and third terminals and communicate with the server concurrently.

Example 4' (TCPSocket-5): This is an equivalent implementation of TCP server (concurrent) and client as Example 4, where we use the Thread class to implement multi-threading in the server program (https://docs.python.org/3/library/threading.html#threading.Thread). Read more about multithreading programming with Python at https://www.tutorialspoint.com/python/python_multithreading.htm.

Besides, the server program runs starting from the following code:

```
if __name__ == '__main__':
        server = ServerMain()
        server.server_run()
```

The purpose of the line of code if __name__ == '__main__' is to tell whether the current module is read directly by the Python interpreter, i.e., whether your program is run as the main program (read more at https://stackoverflow.com/questions/419163/what-does-if-name-main-do). ServerMain() is to create an instance of the ServerMain class, and then we run the method server_run() defined in the ServerMain class.

Test the programs following steps given in Example 4.

Example 5 (UDPSocket): This is the UDP server and client example in the lecture slides.

- **Launch one terminal** and switch to the directory of UDPSocket. Run the server program as follows:

```
python3 UDPServer.py
```

You will see the prompt in the terminal "The server is ready to receive".

- **Launch another terminal** and switch to the directory of UDPSocket. Run the client program as follows:

```
python3 UDPClient.py
```

You will see the prompt "Input a lowercase sentence:". You can enter such a sentence and press "enter". Then you will see an uppercase sentence returned from the server.

Lab Exercise: Simple File Transfer Client and Server

We now implement a simple client/server application for file transfer, where the server receives a file that the client sends.

Step 1: Download **lab1_materials.zip** from Moodle. Unzip it and you will find two files provided: server/FTServer.py and client/FTClient.py. Copy a file into the client folder which your client program will send to the server.

Step 2: Open **FTClient.py** using a text editor, which contains the complete implementation of the client program. Study the client program carefully and you will learn from its code to complete the server program.

a. The client is to be started by command "python3 FTClient.py <Server_addr> <Server_port> <filename>". For example, if your server program is running on the same computer at port 12345 and you are sending file socketprogramming.pdf from the client to the server, the command to type on your terminal to start the client program is python3 FTClient.py 127.0.0.1 12345 socketprogramming.pdf.

Check out the following code at the end of the client program first :

```
if __name__ == '__main__':
        if len(sys.argv) != 4:
                print("Usage: python3 FTClient.py <Server_addr> <Server_port> <filename>")
                sys.exit(1)
        main(sys.argv)
```

The purpose of the line of code if __name__ == '__main__' is to tell whether the current module is read directly by the Python interpreter, i.e., whether your program is run as the main program (read more at https://stackoverflow.com/questions/419163/what-does-if-name-main-do). sys.argv is the list of command-line arguments, i.e., FTClient.py, 127.0.0.1, 12345, and socketprogramming.pdf following python3 in the command you use to run the client program. sys.argv is passed to the main function as the argument.

b. In the main function, the client program first sends the server the name and size of the file to be sent to the server; then upon "OK" acknowledgement from the server, it reads the file contents from the file system and sends them to the server.

- the getsize function in the os.path module checks whether a file exists and gets its file size (https://docs.python.org/3.8/library/os.path.html).
- The Python built-in function open() is used to open a file for reading. Check out the meaning of the second argument, mode, at https://docs.python.org/3/library/functions.html#open.
- In line 21, the Python built-in function int() is used to convert a string to an integer.
- In line 33, read more about Python string encode() at https://docs.python.org/3/library/stdtypes.html#str.encode.
- In line 38, b"OK" is a Python bytes literal ; using the prefix b the string "OK" becomes a bytes object (https://docs.python.org/3/reference/lexical_analysis.html#literals).
- Read more about file read, write and close at https://docs.python.org/3/tutorial/inputoutput.html.

Step 3: Open **FTServer.py** using a text editor and you will find that it provides a sketch of server program. Implement the server program following the hints given as "#....", to achieve the following service: when the server receives the message containing file name and size from the client, it sends "OK" acknowledgement back to the client; then it creates a file (you

can create the file in the server folder) using the received file name; after that, it receives contents of the file from the client and writes received contents into the file.

Implement exception handling for socket APIs bind, accept, recv, and file open and write calls. Read more about using open to create a file and write to write into a file at https://docs.python.org/3/tutorial/inputoutput.html.

The server is to be started by "python3 FTServer <Server_port>", e.g., python3 FTServer.py 12345.

Step 4: test your programs as follows:

- **Launch one terminal** and switch to the directory of Lab1/server. Run the server program as follows:

```
python3 FTServer.py 12345
```

- **Launch the second terminal** and switch to the directory of Lab1/client. Run the client program as follows:

```
python3 FTClient.py 127.0.0.1 12345 filename
```

Here, filename is the name of the file which you copy into the client folder.

Here is a sample output when running the application on the same machine:

Server program:

```
cwu-imac27:server cwu$ python3 FTServer.py 12345
The server is ready to receive
Connection established. Here is the remote peer info: ('127.0.0.1', 52900)
Open a file with name '3_SocketProg_COMP3234_s2020.pdf' with size 1577628 bytes
Start receiving . . .
[Completed]
```

Client program:

```
cwu-imac27:client cwu$ python3 FTClient.py 127.0.0.1 12345 3_SocketProg_COMP3234
_s2020.pdf
Connection established. My socket address is ('127.0.0.1', 52900)
Start sending ...
[Completed]
```

After successfully sending a file from the client to the server, you should see the new file created in the server folder.

Submission:

You should submit the following files in the specified folder structure:
(1) server/FTServer.py
(2) client/FTClient.py
Please compress the above files/folders in a lab1-yourstudentid.zip file and submit it on Moodle before 23:59 Saturday Oct 05, 2024:

(1) Login Moodle.
(2) Find "Lab 1" in the left column and click "Lab 1".
(3) Click "Add submission", browse your .zip file and save it. Done.
(4) You will receive an automatic confirmation email, if the submission was successful.
(5) You can "Edit submission" to your already submitted file, but ONLY before the deadline.