A simple TCP system capable of transfering files

Developer Nguyen Trong Minh
Report writer Nguyen Trong Minh

System design

As mentioned in the assignment, we created a system consists of 2 members, a server and a client. We also created a file to import constants used in our system.

```
HOST = '127.0.0.1'
PORT = 8080
TOTALCLIENTS = 1
BUFFERSIZE = 1024

class Signal(Enum):
    CLOSE_SERVER = b'CLS'
    SEND_A_FILE = b'SAF'
    REQUEST_A_FILE = b'RAF'
    SEND_A_REPO = b'SAR'
    REQUEST_A_REPO = b'RAR'
    DONE = b'DON'
    ERROR = b'ERR'
    PING = b'PIN'
    PONG = b'PON'

SIGNALSIZE = getsizeof(Signal.CLOSE_SERVER.value)
```

The code above also revealed functions exists in our system that we will go into further detail later. For the system design, you can see that we created Signal class to store signal used in communication between server and client. We introduced 5 signals for noticing server about what is going to happen from client side `CLOSE_SERVER`, `SEND_A_FILE`, `SEND_A_REPO`, `REQUEST_ A_FILE`, `REQUEST_A_REPO` and 4 signals to communicate between server and client `DONE`, `ERROR`, `PING`, `PONG`. We also fixed size of each bytes buffer to be sent for files' content at 1024 bytes, and for signal is the size of 36 bytes.

Our server first bind itself to a port, in our case 127.0.0.1:8080, it then start listening to any connection from client, which is 1 only.

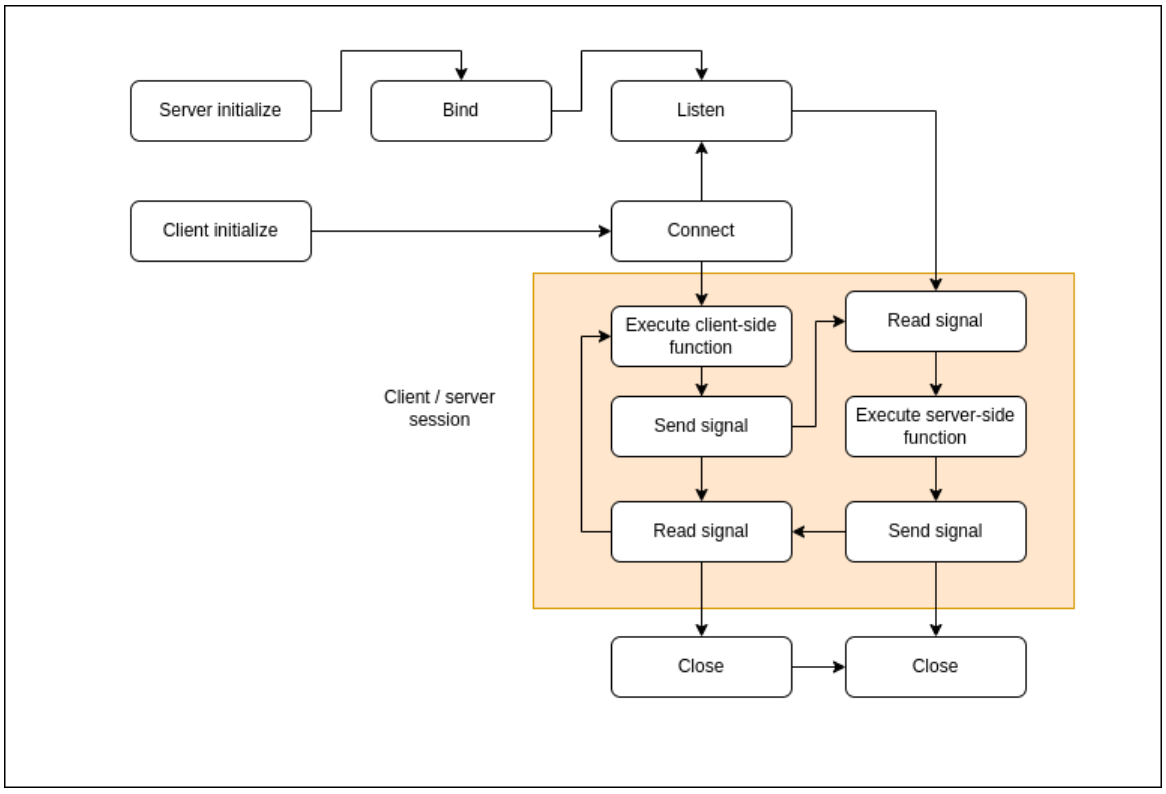
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((HOST, PORT))
sock.listen(TOTALCLIENTS)

# Establishing Connections
client_socket, clar = sock.accept()
```

At this point we used a loop to check if the client has sent any signal to the server to send or request any file. We have 4 functions to send and receive any file as requseted from client. If the signal sent from client match specific signal addressed for each function, the server executes it.

```
while True:
  sig = receive_signal(client_socket)
  if sig == Signal.CLOSE_SERVER.value:
      print('closing')
      break
  if sig == Signal.SEND_A_FILE.value:
      receive_file(client_socket, file_name=r'file-transfered/server-receive/recv0.txt')
  if sig == Signal.REQUEST_A_FILE.value:
      file_name = client_socket.recv(BUFFERSIZE)
      send_file(client_socket, file_name.decode())
  if sig == Signal.SEND_A_REPO.value:
      receive_repo(client_socket, r'file-transfered/server-receive/')
  if sig == Signal.REQUEST_A_REPO.value:
      repo_name = client_socket.recv(BUFFERSIZE)
      send_repo(client_socket, repo_name.decode())
  if not sig:
      break
  print(sig.decode())
```

With the client, it is simply a sequential code to execute specific task that we defined inside Function desgin. After all, take a look at our server workflow graph:



Functions design

In this section, we will go through 4 functions designed to send 1 or more files between server and client in our system. The first funtion is to send a single file from client to server:

```
def send_file(client_socket : socket.socket, file_name: str):
  """Send a file to server"""
  if not exists(file_name):
      raise FileNotFoundError(f'{file_name} does not exist')

  send_signal(client_socket, Signal.SEND_A_FILE )
  sleep(0.1)

  client_socket.send(file_name.split("/")[-1].encode())
  sleep(0.1)

  with open(file_name, 'rb') as f:
      while chunk := f.read(BUFFERSIZE):
          client_socket.send(chunk)

  sleep(0.1)
  send_signal(client_socket, Signal.DONE)
```

Other works

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.