Computer Engineering 4DN4
Laboratory 3
Online Grade Retrieval Application

Group 11
Hengbo Huang - 400241747
Yinwen Xu - 400195279

Lab Contribution: Hengbo Huang : client part
Yinwen Xu : server part

Server Part:

```
def Server_list(self):
    dir_path = "/Users/ashenone/Desktop"
    dir_contents = os.listdir(dir_path)
    print("This is a file sharing directory listing", dir_contents)
```

For the server part, this function will print all the local files in the server which can be downloaded to the client.

```
ALL_IF_ADDRESS = "0.0.0.0"
SERVICE_SCAN_PORT = 30001
ADDRESS_PORT = (ALL_IF_ADDRESS, SERVICE_SCAN_PORT)

SCAN_CMD = "SERVICE DISCOVERY"
SCAN_CMD_ENCODED = SCAN_CMD.encode(MSG_ENCODING)

MSG = "Xu's File Sharing Service"
MSG_ENCODED = MSG.encode(MSG_ENCODING)

PORT = 50000
RECV_SIZE = 1024
BACKLOG = 5

CMD_FIELD_LEN           = 1 # 1 byte commands sent from the client.
FILENAME_SIZE_FIELD_LEN  = 1 # 1 byte file name size field.
FILESIZE_FIELD_LEN       = 8 # 8 byte file size field.

# Define a dictionary of commands. The actual command field value must
# be a 1-byte integer. For now, we only define the "GET" command,
# which tells the server to send a file.

CMD = {
       "scan" : 1,
       "Connect" : 2,
       "llist" : 3,
       "rlist" : 4,
       "put" : 5,
       "get" : 6,
       "bye" : 7
}

MSG_ENCODING = "utf-8"
SOCKET_TIMEOUT = 100

CMD_list = [CMD["scan"],CMD["Connect"],CMD["llist"],CMD["rlist"],CMD["put"],CMD["get"],CMD["bye"]]

command_list = ["scan","Connect","llist","rlist","put","get","bye"]
```

These are some initialization variables for the server part.

```
def select_protocal(self):

    self.create_socket()
    self.create_listen_socket()

    poll = select.poll()
    poll.register(self.socket, select.POLLIN)
    poll.register(self.UDP_socket, select.POLLIN)

    while True:
        events = poll.poll()

        for fd, event in events:
            if fd == self.UDP_socket.fileno():

                self.receive_forever()
            elif fd == self.socket.fileno():

                self.process_connections_forever()
```

In this part, we use native poll to make sure our server can listen to both UDP connection and TCP connection.

```
def create_socket(self):
    try:
        # Create an IPv4 UDP socket.
        self.UDP_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        # Get socket layer socket options.
        self.UDP_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        # Bind socket to socket address, i.e., IP address and port.
        self.UDP_socket.bind( (Server.HOSTNAME, Server.SERVICE_SCAN_PORT) )

        self.UDP_socket.setblocking(False)

    except Exception as msg:
        print(msg)
        sys.exit(1)

def receive_forever(self):
    while True:
        try:
            print(Server.MSG, "listening on port {} ...".format(Server.SERVICE_SCAN_PORT))
            recvd_bytes, address = self.UDP_socket.recvfrom(Server.RECV_SIZE)

            print("Received: ", recvd_bytes.decode('utf-8'), " Address:", address)

            # Decode the received bytes back into strings.
            recvd_str = recvd_bytes.decode(MSG_ENCODING)

            # Check if the received packet contains a service scan
            # command.
            if Server.SCAN_CMD in recvd_str:
                # Send the service advertisement message back to
                # the client.
                self.UDP_socket.sendto(Server.MSG_ENCODED, address)

                time.sleep(5)
                self.UDP_socket.close()
        except KeyboardInterrupt:
            print()
            sys.exit(1)
        except BlockingIOError:
            print("Waiting for the SCAN!")
            time.sleep(5)
        except OSError:
            self.select_protocal()
```

For this part , when we receive a sacn command which contain the message “SERVICE DISCOVERY”, our server will send the name of the file sharing service and the address as well as the port.

```
ashenone@AshendeMacBook-Air Desktop % python3 Lab3_s.py -r server
This is a file sharing directory listing ['file.txt', 'WechatIMG44.jpeg', 'Wecha
tIMG45.jpeg', '.DS_Store', 'Client_Local_List', '.localized', 'service_announcem
ent.py', 'Lab3_s.py', 'PASSPORT.pdf', 'Screenshot 2023-03-25 at 12.12.47 AM.png'
, 'PASSPORT.jpeg', 'file_download_protocol.py', 'file.doc', 'example.py', 'downl
oad_from_client.txt', '~$file.doc', 'Ebay_object.jpeg', 'Lab3_c3.0.py', 'broadca
st_send_receive.py']
Listening for service discovery messages on SDP on port 30001 ...
Listening for file sharing connections on port 50000 ...
Xu's File Sharing Service listening on port 30001 ...
Received:  SERVICE DISCOVERY  Address: ('10.0.0.79', 57573)
```

```
ashenone@AshendeMacBook-Air Client_Local_List % python3 Lab3_c.py -r client
Command list: get, scan, connect, llist, rlist, put, bye
Command: scan
Sending broadcast scan 0
Sent successfully!
Sending broadcast scan 1
Sent successfully!
Sending broadcast scan 2
Sent successfully!
("Xu's File Sharing Service", ('10.0.0.79', 30001))
```

And also, it will listen to TCP connection. Once a TCP connection received, it will create a socket and analyse the command. We use a if statement to deal with different situations.

```
def get_command(self,connection):

    ##################################################################
    # Process a connection and see if the client wants a file that
    # we have.

    # Read the command and see if it is a GET command.
    status, cmd_field = recv_bytes(connection, CMD_FIELD_LEN)
    # If the read fails, give up.
    #if not status:
        #print("not status, Closing connection ...")
       # connection.close()
       # return
    # Convert the command to our native byte order.
    if cmd_field != "":
        cmd = int.from_bytes(cmd_field, byteorder='big')
        # Give up if we don't get a GET command.
        if cmd not in CMD_list:
            print("Received cmd is :" ,cmd)
            print("Valid command not received. Closing connection ...")
            connection.close()
            return

        if cmd == CMD["rlist"]:
            print("start showing the list from server")
            self.Receive_Rlist(connection)
            print("Finish showing the list")

        elif cmd == CMD["get"]:
            self.Receive_Get(connection)
            print("finish transmitting file to client.")

        elif cmd == CMD["put"]:
            print("start downloading from client.")
            self.Receive_Put(connection)
            print("finish downloading from client.")
        elif cmd == CMD["bye"]:
            print("Receiving bye")
            connection.close()
            print("Close the connection")
```

```
def Receive_Rlist(self,connection):
    try:
        dir_path = "/Users/ashenone/Desktop"
        dir_contents = os.listdir(dir_path)
        print("This is a file sharing directory listing", dir_contents)
        dir_contents_bytes = "\n".join(dir_contents).encode(MSG_ENCODING)
        connection.sendall(dir_contents_bytes)
         print("Connected to server",self.address,"on port",self.port)
         self.connect_to_server()
         self.socket.close()
    except OSError:
        self.select_protocal()
```

This part is for rlist command, it can get the server's local file and make the name of these files into a list. After that, it will convert the list of files into string and encode them into bytes. In this way, it can send through the TCP connections.

```
def Receive_Get(self,connection):

    # GET command is good. Read the filename size (bytes).
    status, filename_size_field = recv_bytes(connection, FILENAME_SIZE_FIELD_LEN)
    if not status:
        print("not status2, Closing connection ...")
        connection.close()
        return
    filename_size_bytes = int.from_bytes(filename_size_field, byteorder='big')
    if not filename_size_bytes:
        print("Connection is closed!")
        connection.close()
        return

    print('Filename size (bytes) = ', filename_size_bytes)

    # Now read and decode the requested filename.
    status, filename_bytes = recv_bytes(connection, filename_size_bytes)
    if not status:
        print("not status3, Closing connection ...")
        connection.close()
        return
    if not filename_bytes:
        print("Connection is closed!")
        connection.close()
        return

    filename = filename_bytes.decode(MSG_ENCODING)
    print('Requested filename = ', filename)
```

```
#################################################################
# See if we can open the requested file. If so, send it.

# If we can't find the requested file, shutdown the connection
# and wait for someone else.
try:
    file = open(filename, 'r').read()
except FileNotFoundError:
    print(Server.FILE_NOT_FOUND_MSG)
    connection.close()
    return

# Encode the file contents into bytes, record its size and
# generate the file size field used for transmission.
file_bytes = file.encode(MSG_ENCODING)
file_size_bytes = len(file_bytes)
file_size_field = file_size_bytes.to_bytes(FILESIZE_FIELD_LEN, byteorder='big')

# Create the packet to be sent with the header field.
pkt = file_size_field + file_bytes

try:
    # Send the packet to the connected client.
    connection.sendall(pkt)
    print("Sending file: ", filename)
    print("file size field: ", file_size_field.hex(), "\n")
     print("Connected to server",self.address,"on port",self.port)
     self.connect_to_server()
    # time.sleep(20)
     self.socket.close()
except socket.error:
    # If the client has closed the connection, close the
    # socket on this end.
    print("Closing client connection ...")
    connection.close()
    return


except OSError:
    self.select_protocal()
```

This part is the code for receiving Get command. Firstly, it will receive the file name size and decode it from bytes to int. After that, it will use the number of bytes to receive the filename bytes and decode it from bytes to string. Once we get the filename, we use open to get the full content of the file if it exists in server. Once it succeeded, we start to use encode and to_bytes to convert the file size and file contents into bytes and send them through TCP connection.

```
def Receive_Put(self,connection):

    # Read the file size field returned by the server.
    print("receiving filename_size_bytes")
    status, filename_size_bytes = recv_bytes(connection, FILENAME_SIZE_FIELD_LEN)
    print(f"Received {len(filename_size_bytes)} bytes of data from client.")

    if not status:
        print("Closing connection ...")
        self.socket.close()
        return

    print("File name size bytes = ", filename_size_bytes.hex())
    if len(filename_size_bytes) == 0:
        self.socket.close()
        return

    filename_size = int.from_bytes(filename_size_bytes,byteorder = 'big')

    print("receiving filename_bytes")
    status, filename_bytes = recv_bytes(connection, filename_size )
    print(f"Received {len(filename_bytes)} bytes of data from client.")

    if not status:
        print("Closing connection ...")
        self.socket.close()
        return

    self.filename = filename_bytes.decode(MSG_ENCODING)

    print("receiving file_size_bytes")
    status, file_size_bytes = recv_bytes(connection, FILESIZE_FIELD_LEN)
    print(f"Received {len(file_size_bytes)} bytes of data from client.")

    if not status:
        print("Closing connection ...")
        self.socket.close()
        return

    print("File size bytes = ", file_size_bytes.hex())
    if len(file_size_bytes) == 0:
        self.socket.close()
        return
```

```
        # Make sure that you interpret it in host byte order.
        file_size = int.from_bytes(file_size_bytes, byteorder='big')
        print("File name size = ", file_size)

        # self.socket.settimeout(4)
        status, recvd_bytes_total = recv_bytes(connection, file_size)
        if not status:
            print("Closing connection ...")
            self.socket.close()
            return
        # print("recvd_bytes_total = ", recvd_bytes_total)
        # Receive the file itself.
        try:
            # Create a file using the received filename and store the
            # data.
            print("Received {} bytes. Creating file: {}" \
                  .format(len(recvd_bytes_total), self.UPLOAD_FILE_NAME))

            with open(self.UPLOAD_FILE_NAME, 'w') as f:
                recvd_file = recvd_bytes_total.decode(MSG_ENCODING)
                f.write(recvd_file)
                self.SERVER_LIST.append(self.UPLOAD_FILE_NAME)
            print(recvd_file)
             print("Connected to server",self.address,"on port",self.port)
             self.connect_to_server()
             self.socket.close()
        except KeyboardInterrupt:
            print()
            exit(1)
        except OSError:
            self.select_protocal()

    def Receive_Rlist(self,connection):
        try:
            dir_path = "/Users/ashenone/Desktop"
            dir_contents = os.listdir(dir_path)
            print("This is a file sharing directory listing", dir_contents)
            dir_contents_bytes = "\n".join(dir_contents).encode(MSG_ENCODING)
            connection.sendall(dir_contents_bytes)
             print("Connected to server",self.address,"on port",self.port)
             self.connect_to_server()
             self.socket.close()
        except OSError:
            self.select_protocal()
```

This part is the code when we receive a put command. Firstly, it will read 1 byte to get the file name size in bytes. After this, it will decode into int and use the number to receive filename and decode it into string. After that, it will read 8 bytes to get the file size in bytes. Next, it will convert it from bytes to int and use the number to receive the full file contents. After receiving all the bytes, it will use a with open function to create a file and write everything we get into the file.

Client Part :

1. **Get command:**

```python
def __init__(self):

    self.send_console_input_forever()

def get_console_input(self): ···

def send_console_input_forever(self):
    while True:
        try:
            print("Command list: get, scan, connect, llist, rlist, put, bye")
            self.command = input("Command: ")
            if self.command != '':
                self.get_cmd()


        except (KeyboardInterrupt, EOFError):
            print()
            print("Closing client socket ...")
            self.socket.close()
            sys.exit(1)
```

```python
def get_cmd(self):
    if self.command == "get":
        self.get_file()
       # print("Connected to server",self.address,"on port"
        self.connect_to_server()
    elif self.command == "scan":
        self.get_soket_broadcasts()
        self.scan_for_service()
    elif self.command == "connect":
        print("Please enter IP address and Port number")
        self.address = str(input("IP address: "))
        self.port = int(input("Port number: "))
        self.address_port = (self.address, self.port)
        self.connect_to_server()
    elif self.command == "llist":
        self.get_local_list()
    elif self.command == "bye":
        self.socket.close()
        print(" Close the current  server connection.")
    elif self.command == "put":
        self.put_file()
        self.connect_to_server()
    elif self.command == "rlist":
        self.get_remote_list()
        self.connect_to_server()
    else:
        print ("Unknown command, please enter again")
        self.send_console_input_forever()
```

When the client part is called , first user needs to enter the command, there is a print command list to prompt the user. Also if the enter command is not in the list, it will let user to enter the command again.

2. **Scan function:**
   The client transmitsIP address 255.255.255.255 on port 30001 and listens for the server's response. The get_soket function for broadcast and other functions are different.

```
def get_soket_broadcasts(self):
    try:
        # Service discovery done using UDP packets.
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        # Arrange to send a broadcast service discovery packet.
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        # Set the socket for a socket.timeout if a scanning recv
        # fails.
        self.socket.settimeout(Client.SCAN_TIMEOUT);
    except Exception as msg:
        print(msg)
        sys.exit(1)
```

3. **Connect function**

   The user can enter the ip address and port number in the terminal.

```
def get_socket(self):
    try:
        # Service discovery done using UDP packets.
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    except Exception as msg:
        print(msg)
        exit()
```

```
def connect_to_server(self):
    try:
        self.socket.connect((Server.HOSTNAME, Server.PORT))
        if(self.address == ""):
            print("Please enter ip adress and port number")
            self.address = str(input("IP address: "))
            self.port = int(input("Port number: "))
            self.connect_to_server()
        #elif (self.address == "255.255.255.255" ):
            #self.get_soket_broadcasts()
            #self.socket.connect((self.address, self.port))
        else:
            self.get_socket()
            self.socket.connect((self.address, self.port))
        print("Connected to server",self.address,"on port",self.port )
    except Exception as msg:
        print(msg)
        exit()
```

4. **Local List function :**

   For the llist function, all the file which is in the same folder will be print. We separate the folder of server and client so the rlist and llist function will not mix up files.

```
def get_local_list(self):
    dir_path = "/Users/ashenone/Desktop/Client_Local_List"
    dir_contents = os.listdir(dir_path)
    print( "Local List :")
    for i in dir_contents :
        print(i)
```

5. **Remote list function :**
   Client will receive the list of remote file list and print it out.

```
def get_remote_list(self):
    cmd_field = CMD["rlist"].to_bytes(CMD_FIELD_LEN, byteorder='big')
    self.socket.sendall(cmd_field)
    print("Sending to {} ...".format(self.address_port))
    self.remote_list_str = self.socket.recv(Client.RECV_SIZE).decode(MSG_ENCODING)
    self.remote_list = self.remote_list_str.split("\n")

    print("Content of this file sharing directory listing: ")
    for i in self.remote_list:
        print(i)
```

6. **Get function**
   The get function is similar to the code provided by the professor.The user can enter the file name in the terminal to get the file he/she wants.

```
# Create the packet filename field.
#filename_field_bytes = Server.REMOTE_FILE_NAME.encode(MSG_ENCODING)
self.filename = input("Filename:")
filename_field_bytes = self.filename.encode(MSG_ENCODING)
# Create the packet filename size field.
filename_size_field = len(filename_field_bytes).to_bytes(FILENAME_SIZE_FIELD_LEN, byteorder='big')

# Create the packet.
print("CMD field: ", cmd_field.hex())
print("Filename_size_field: ", filename_size_field.hex())
print("Filename field: ", filename_field_bytes.hex())

pkt = cmd_field + filename_size_field + filename_field_bytes
```

Also, since the user needs to take time to enter the name of the file, the timeout value should be larger.

```
CMD = {"scan" : 1 ,
       "get"  : 2,
       "list" : 3,
       "put"  : 4}

MSG_ENCODING = "utf-8"
SOCKET_TIMEOUT = 100
```

7. **Put function**

   Put function is similar to the get function. The put function is similar to the action of the server in the get function, which is received file name and send the file package to the client. So the code of Client's put function is written based on the server's get function.

```
def put_file(self):
    cmd_field = CMD["put"].to_bytes(CMD_FIELD_LEN, byteorder='big')
    self.filename = input("Filename:")

    try:
        file = open(self.filename, 'r').read()
    except FileNotFoundError:
        print(Server.FILE_NOT_FOUND_MSG)
        self.socket.close()
        return

    filename_bytes = self.filename.encode(MSG_ENCODING)
    filename_size_field = len(filename_bytes).to_bytes(FILENAME_SIZE_FIELD_LEN, byteorder='big')
    file_bytes = file.encode(MSG_ENCODING)
    file_size_bytes = len(file_bytes)
    file_size_field = file_size_bytes.to_bytes(FILESIZE_FIELD_LEN, byteorder='big')
    print("CMD field: ", cmd_field.hex())
    print("Filename_size_field: ", file_size_field.hex())
    print("Filename field: ", file_bytes.hex())
    pkt = cmd_field + filename_size_field + filename_bytes +file_size_field + file_bytes
    try:
        # Send the packet to the connected client.
        self.socket.sendall(pkt)
        print(f"Sent {len(pkt)} bytes of data to server.")
        print("Sending file: ", self.filename)
        print("file size field: ", file_size_field.hex(), "\n")
        self.TRANS_LIST.append("put " + self.filename)
        # time.sleep(20)
    except socket.error:
        # If the client has closed the connection, close the
        # socket on this end.
        print("Closing client connection ...")
        self.socket.close()
        return
    finally:
        self.socket.close()
        return
```

8. **Bye function**

```
elif self.command == "bye":
    self.socket.close()
    print(" Close the current  server connection.")
```

Some examples to show it works

```
[ashenone@AshendeMacBook-Air Desktop % python3 Lab3_s.py -r server
This is a file sharing directory listing ['file.txt', 'WechatIMG44.jpeg', 'Wecl
tIMG45.jpeg', '.DS_Store', 'Client_Local_List', '.localized', 'service_announc
ent.py', 'Lab3_s.py', 'PASSPORT.pdf', 'Screenshot 2023-03-25 at 12.12.47 AM.pn
, 'PASSPORT.jpeg', 'file_download_protocol.py', 'file.doc', 'example.py', 'dow
oad_from_client.txt', '~$file.doc', 'Ebay_object.jpeg', 'Lab3_c3.0.py', 'broad
st_send_receive.py']
Listening for service discovery messages on SDP on port 30001 ...
Listening for file sharing connections on port 50000 ...
```

Scan:

```
ashenone@AshendeMacBook-Air Desktop % python3 Lab3_s.py -r server
This is a file sharing directory listing ['file.txt', 'WechatIMG44.jpeg', 'Wec
tIMG45.jpeg', '.DS_Store', 'Client_Local_List', '.localized', 'service_announc
ent.py', 'Lab3_s.py', 'PASSPORT.pdf', 'Screenshot 2023-03-25 at 12.12.47 AM.pn
, 'PASSPORT.jpeg', 'file_download_protocol.py', 'file.doc', 'example.py', 'dow
oad_from_client.txt', '~$file.doc', 'Ebay_object.jpeg', 'Lab3_c3.0.py', 'broad
st_send_receive.py']
Listening for service discovery messages on SDP on port 30001 ...
Listening for file sharing connections on port 50000 ...
Xu's File Sharing Service listening on port 30001 ...
Received:  SERVICE DISCOVERY  Address: ('10.0.0.79', 65242)
Xu's File Sharing Service listening on port 30001 ...
Listening for service discovery messages on SDP on port 30001 ...
Listening for file sharing connections on port 50000 ...
```

```
AttributeError: 'Client' object has no attribute 'socket'
ashenone@AshendeMacBook-Air Client_Local_List % python3 Lab3_c.py -r client
Command list: get, scan, connect, llist, rlist, put, bye
Command: scan
Sending broadcast scan 0
Sent successfully!
Sending broadcast scan 1
Sent successfully!
Sending broadcast scan 2
Sent successfully!
("Xu's File Sharing Service", ('10.0.0.79', 30001))
Command list: get, scan, connect, llist, rlist, put, bye
```

rlist:

```
Command list: get, scan, connect, llist, rlist, put, bye
Command: llist
Local List :
download_from_server.txt
.DS_Store
upload_file2.txt
upload_file.txt
Lab3_c.py
```

Connect

```
Connection received from ('127.0.0.1', 59483).
Command list: get, scan, connect, llist, rlist, put, bye
Command: connect
Please enter IP address and Port number
IP address: 127.0.0.1
Port number: 50000
Connected to server 127.0.0.1 on port 50000
Command list: get, scan, connect, llist, rlist, put, bye
```

Rlist

```
start showing the list from server
This is a file sharing directory listing ['file.txt', 'WechatIMG44.jpeg', 'Wecha
tIMG45.jpeg', '.DS_Store', 'Client_Local_List', '.localized', 'service_announcem
ent.py', 'Lab3_s.py', 'PASSPORT.pdf', 'Screenshot 2023-03-25 at 12.12.47 AM.png'
, 'PASSPORT.jpeg', 'file_download_protocol.py', 'file.doc', 'example.py', 'downl
oad_from_client.txt', '~$file.doc', 'Ebay_object.jpeg', 'Lab3_c3.0.py', 'broadca
st_send_receive.py']
Finish showing the list
Command: rlist
Sending to ('127.0.0.1', 50000) ...
Content of this file sharing directory listing:
file.txt
WechatIMG44.jpeg
WechatIMG45.jpeg
.DS_Store
Client_Local_List
.localized
service_announcement.py
Lab3_s.py
PASSPORT.pdf
Screenshot 2023-03-25 at 12.12.47 AM.png
PASSPORT.jpeg
file_download_protocol.py
file.doc
example.py
download_from_client.txt
~$file.doc
Ebay_object.jpeg
Lab3_c3.0.py
broadcast_send_receive.py
```

Get:

```
Command list: get, scan, connect, llist, rlist, put, bye
Command: get
Filename:file.txt
CMD field:  06
Filename_size_field:  08
Filename field:  66696c652e747874
Sending to ('127.0.0.1', 50000) ...
File size bytes =  000000000000001e
File size =  30
Received 30 bytes. Creating file: download_from_server.txt
123125789461932765891asjkfhgas
Connected to server 127.0.0.1 on port 50000
Command list: get, scan, connect, llist, rlist, put, bye
--------------------------------------------------------------------------
Connection received from ('127.0.0.1', 60615).
Filename size (bytes) =  8
Requested filename =  file.txt
Sending file:  file.txt
file size field:  000000000000001e

finish transmitting file to client.
Waiting for the connection!
```

Put:

```
Command list: get, scan, connect, llist, rlist, put, bye
Command: put
Filename:upload_file.txt
CMD field:  05
Filename_size_field:  000000000000002f
Filename field:  492077616e7420746f2075706c6f6164207468697320666696c6520616e64206
9742073756363657366756c6c7921
Sent 72 bytes of data to server.
Sending file:  upload_file.txt
file size field:  000000000000002f
```

```
Connection received from ('127.0.0.1', 61060).
start downloading from client.
receiving filename_size_bytes
Received 1 bytes of data from client.
File name size bytes =  0f
receiving filename_bytes
Received 15 bytes of data from client.
receiving file_size_bytes
Received 8 bytes of data from client.
File size bytes =  000000000000002f
File name size =  47
Received 47 bytes. Creating file: download_from_client.txt
I want to upload this file and it successfully!
finish downloading from client.
--------------------------------------------------------------------------
```