Facultatea Calculatoare, Informatica si Microelectronica

Universitatea Tehnica a Moldovei

Programarea retelelor

Lucrarea de laborator#5

TCP

Autor:
Ana Sutreac

lector asistent:
Alexandru Gavrisco
lector superior:
Dumitru Ciorba

Laboratory work #5

1 Scopul lucrarii de laborator

Elaborarea unei aplicații Client - Server cu scopul studierii protocolului de la nivelul de transport - TCP.

2 Obiective

- Studierea nivelului de transport în rețea și TCP/IP;
- Studierea BSD sockets API;
- Elaborarea unei aplicații client-server.

3 Laboratory work implementation

3.1 Sarcina de baza (5 - 7)

Primul pas logic este să studiați interfața oferită de limbaj pentru lucru cu BSD sockets.

Scopul este să implementați o aplicație client-server, deci următorul pas este stabilirea protocolului de comunicare între client și server.

Formatul mesajelor

Pentru a ușura acest proces, se stabilește următorul format al mesajelor:

- Comenzile de la client încep cu /

- Numele comenzii poate conține A-Za-z0-9_ De exemplu: /help

- Daca comanda acceptă parametri, atunci după comanda urmeaza spațiu si restul datelor. Exemplu: /hello John

- Dacă serverul primește o comandă invalidă - se răspunde cu un mesaj informativ.

Protocolul de comunicare

Aceasta și este prima sarcină - să descrieți protocolul de comunicare între client și server. Acest document trebuie păstrat în repozitoriu și inclus în raport. Documentul trebuie să fie plain-text, se recomandă utilizarea markdown.

Documetul cu specificația protocolului trebuie să conțină:

Formatul mesajelor

- Comenzile suportate de server

- Exemple de răspuns la fiecare comandă

- Comenzile acceptate de server

Comenzile obligatorii care trebuie să le implementeze serverul:

- /help - răspunde cu o listă a comenzilor suportate și o descriere a fiecărei comenzi; - /hello Text - raspunde cu textul care a fost expediat ca paremetru¡ - alte 3 comenzi cu funcțional diferit (e.g. timpul curent, generator de cifre, flip the coin etc)

Cerințe pentru sistem

Cerințele de bază pentru aplicație sunt:

- O aplicație client care se conectează la server și permite transmiterea comenzilor;

— Comenzile sunt introduse de utilizator de la tastatură;

— Răspunsul primit de la server este afișat utilizatorului.

-O aplicație server care:

— Acceptă conexiunea de la client la un careva port;

— Primește comenzile de la client;

— Transmite un răspuns clientului.

Constrîngeri:

Să se utilizeze doar interfața BSD sockets oferită de limbaj/platformă.

3.2 Analiza lucrarii de laborator

https://github.com/anashutreac/PR-labs/tree/master/lab_5

3.3 Anexa 1

Protocolul de comunicare intre client si server

Formatul mesajelor

- Comenzile de la client încep cu /

- Numele comenzii poate contine A-Za-z0-_ De exemplu: /help

- Daca comanda accepta parametri, atunci dupa comanda urmeaza spatiu si restul datelor. Exemplu: /hello John

- Daca serverul primeste o comanda invalida - se raspunde cu un mesaj informativ.

Comenzile suportate de server

* /dice - virtually role the dice

* /flip - virtually flip a coin

* /help - short usage description of the

* /hello - hello Commander. Parameter : name

* /uptime - displays current system time

* /shut_down - shut down socket server, use carefully

Exemple de raspuns la fiecare comanda

* /dice - 5

* /flip - Tails

* /help -

/flip - virtually flip a coin

/uptime - displays current system time

/hello - hello Commander. Parameter : name

/dice - virtually role the dice

/help - short usage description of the command

/shut_down - shut down socket server, use carefully

* /hello Ana - Hello Ana! Nice to have a connection with you.

* /uptime - 23:32:53.396670

* /shut_down - You finished the server session. Further commands wont́ work!

3.4 Anexa 2

client.py

```
import socket

# We're using TCP/IP as transport
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the given `address` and `port`
server_address = ('127.0.0.1', 8080)
print('Connecting to 127.0.0.1:8080')
client_socket.connect(server_address)
try:
        # Read input from the user (as string)
        while 1:
                print('Enter a command:')
                data = input('>>')
                client_socket.sendall(data.encode())
                # Receive 1kB of data from the server
                data = client_socket.recv(1024)
                print('<<<:' + data.decode() + '\n')
finally:
        print('Closing socket')
        server_address.close()
```

server.py

```
import datetime
import socket
import threading
import random


def help_info(arrg):
        info = ''
        for key, value in command_descriptions.items():
        info += key + ' - ' + value + '\n'
        return info
```

```
def hello(arrg):
        return 'Hello ' + arrg + '! Nice to have a connection with you.'


def dice(arrg):
        return random.choice(['1', '2', '3', '4', '5', '6'])


def uptime(arrg):
        return str(datetime.datetime.now().time())


def flip(arrg):
        return random.choice(['Heads', 'Tails'])


def stop_server(arrg):
        server_socket.close()
        print('You finished the server session')
        return 'You finished the server session. Further commands won\'t work


def handle_commands(command, param):
        try:
                func = commands.get(command)
                if func is None:
                        raise KeyError
                print(command, param)
                return func(param)
        except KeyError:
                return 'Wrong command Commander! Try once again or help!'


commands = {
'/dice': dice,
'/flip': flip,
'/help': help_info,
'/hello': hello,
'/uptime': uptime,
'/shut_down': stop_server
```

```
}


command_descriptions = {
'/dice': '- virtually role the dice',
'/flip': '- virtually flip a coin',
'/help': ' - short usage description of the command',
'/hello': '- hello Commander. Parameter : name',
'/uptime': '- displays current system time',
'/shut_down': '- shut down socket server, use carefully'

}

# We're using TCP/IP as transport
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# Bind to the given address and port
server_socket.bind(('127.0.0.1', 8080))


# Start listening on socket, maximum number of queued connections - 10
server_socket.listen(5)
print('Socket created successfully')


def process(client_message):
        param = ''
        message_arrg = client_message.strip().split(' ')
        if len(message_arrg) == 2:
                command, param = message_arrg
        elif len(message_arrg) == 1:
                command = message_arrg[0]
        else:
                return "Wrong command Commander! Your message should have the
        return handle_commands(command, param)


# Function for handling connections. This will be used to create threads
def handle_client(connection):
# infinite loop so that function do not terminate and thread do not end.
```

```
while True:

        # Receiving from client
        data = connection.recv(1024)
        reply = process(data.decode()).encode()
        if not data:
                break

                connection.sendall(reply)

        # came out of loop
        connection.close()


# now keep talking with the client
while 1:
        # wait to accept a connection - blocking call
        connection, address = server_socket.accept()
        print('Connected with ' + address[0] + ':' + str(address[1]))

        thread = threading.Thread(target=handle_client, args=(connection,))
        thread.start()
```

Concluzie

Aici trebuie sa fie concluzia ta.

References

1 Aldebran Robotics, *official page*, www.aldebaran.com/en

2 Timo Ojala, *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*, 2002

3 Biometric, www.biometricupdate.com/201501/history-of-biometrics