



Códigos - Socket

Cliente e Server TCP

```
##-----CLIENTE-----##

# Importação das funções necessárias do módulo socket
from socket import *

# Definição do endereço e porta do servidor
serverName = 'localhost' # Endereço do servidor (localhost r
serverPort = 12000 # Porta em que o servidor está escutando

# Criação do socket do cliente
clientSocket = socket(AF_INET, SOCK_STREAM) # Criação de um
# Conexão com o servidor
clientSocket.connect((serverName, serverPort)) # Conexão com

# Solicitação de entrada ao usuário
sentence = input('Digite alguma coisa:') # Solicita ao usuár.
```

```

# Envio da frase para o servidor após codificação em UTF-8
clientSocket.sendall(str.encode('utf-8'))

# Recebimento de dados do servidor (até 1024 bytes) e armazenados
dados = clientSocket.recv(1024)

print("Mensagem ecoada:", dados.decode()) #dados recebidos de

# Fechamento do socket do cliente
clientSocket.close() # Libera os recursos do socket após a c

```

```

##-----SERVER-----##

```

```

import time # Importa o módulo de tempo para lidar com atrasos
from socket import * # Importa todas as funções necessárias

```

```

serverPort = 12000 # Define a porta em que o servidor irá es
serverSocket = socket(AF_INET, SOCK_STREAM) # Cria um socket
serverSocket.bind(('', serverPort)) # Associa o socket ao en
serverSocket.listen(1) # Habilita o servidor para aceitar co
print("Aguardando conexão com cliente...")
print("The server is ready to receive") # Imprime uma mensag

```

```

#Agora o servidor precisa aceitar a conexão.Basta fazer:

```

```

while True: # Loop infinito para receber conexões continuame
    conexao, endereco = serverSocket.accept() # Aceita uma
    print("Conectado em:", endereco)

    dados = conexao.recv(1024) # Recebe dados da conexão (a

```

```

# Quando não tiver mais dados para receber
if not dados:
    print("Fechando conexão")
    conexao.close() # Fecha a conexão com o cliente
    break

conexao.sendall(dados) # Envia de volta os dados para o
text = dados.decode('utf-8') # Decodifica os dados recebidos
capitalizedSentence = text.upper() # Converte a frase recebida para maiúsculas
conexao.send(capitalizedSentence.encode('utf-8')) # Envia a frase convertida

```

Cliente e Server UDP

```

##-----CLIENTE-----##

from socket import * # Importa todas as funções necessárias para o cliente

serverName = 'localhost' # Endereço do servidor
serverPort = 12000 # Porta do servidor

# Criação do socket do cliente para comunicação via UDP
clientSocket = socket(AF_INET, SOCK_DGRAM)

# Solicitação de entrada ao usuário
message = input("Input lowercase sentence:") # Solicita ao usuário uma frase

# Envio da mensagem para o servidor
# A função sendto() é usada para enviar a mensagem para o endereço do servidor
clientSocket.sendto(message.encode("utf-8"), (serverName, serverPort))

# Recebimento da mensagem modificada do servidor
# A função recvfrom() é usada para receber dados do servidor
# Retorna uma tupla contendo a mensagem e o endereço do servidor

```

```

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

# Exibição da mensagem modificada recebida do servidor
print(modifiedMessage.decode("utf-8"))

# Fechamento do socket do cliente
clientSocket.close() # Libera os recursos do socket após a c

##-----SERVER-----##

from socket import * # Importa todas as funções necessárias

serverPort = 12000 # Porta em que o servidor irá escutar por

# Criação do socket do servidor para comunicação via UDP
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Associação do socket a um endereço e porta específicos
serverSocket.bind('', serverPort)

print("The server is ready to receive") # Indica que o servi

while True: # Loop infinito para receber mensagens continuam
    # Recebe uma mensagem e o endereço do cliente que a envio
    message, clientAddress = serverSocket.recvfrom(2048)

    # Decodifica a mensagem recebida para UTF-8 e converte pa
    modifiedMessage = message.decode("utf-8").upper()

    # Envia a mensagem modificada de volta para o cliente
    # Utiliza o endereço do cliente recebido anteriormente pa
    serverSocket.sendto(modifiedMessage.encode("utf-8"), clie

```

CODIFICAÇÃO DADOS

- Python lida com dados como sequências de bytes, não como strings diretamente
- Quando você envia dados por meio de um socket, eles precisam ser convertidos em uma representação de bytes para serem transmitidos pela rede.

```
import socket

# Criando um socket TCP/IP
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Endereço e porta do servidor
server_address = ('localhost', 8080)

# Conectando ao servidor
client_socket.connect(server_address)

# Valores de x e y que queremos enviar
x = 42
y = 77

# Construindo a mensagem como uma string
message = f"{x} {y}"

# Codificando a mensagem em bytes usando a codificação ASCII
encoded_message = message.encode('ascii')

# Enviando a mensagem codificada para o servidor
client_socket.sendall(encoded_message)

# Fechando o socket
client_socket.close()
```

Problema no armazenamento de endereços IP e portas como inteiros nas estruturas de dados usadas em programação de rede

- Por exemplo, em uma estrutura `sockaddr_in`, a porta é armazenada como um inteiro de 16 bits (`u_short sin_port`) e o endereço IP é armazenado como um inteiro de 32 bits (`in_addr sin_addr`).
- Alguns sistemas usam a ordem de bytes big-endian, onde os bytes mais significativos (mais à esquerda) são armazenados primeiro, enquanto outros sistemas usam a ordem de bytes little-endian, onde os bytes menos significativos (mais à esquerda) são armazenados primeiro.
- Isso pode criar problemas quando duas máquinas com diferentes ordens de bytes tentam se comunicar pela rede. Por exemplo, se um endereço IP for representado como `128.119.40.12` (big-endian) e for interpretado por uma máquina little-endian, ele pode ser interpretado incorretamente como `12.40.119.128`. Isso pode causar falhas na comunicação.

```
import socket

# Criando um socket UDP para o cliente
client_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Configurando a porta do cliente
client_port = 12345 # Exemplo de porta

# Configurando a mensagem a ser enviada
message = b"Hello, server!" # Mensagem em bytes

# Convertendo a porta do cliente para a ordem de bytes da rede
client_port = socket.htons(client_port)

# Enviando a mensagem para o servidor
server_address = ('localhost', 1111) # Endereço e porta do servidor
client_sock.sendto(message, (server_address[0], server_address[1]))
```

```
# Fechando o socket do cliente
client_sock.close()
```

```
import socket

# Criando um socket UDP para o servidor
server_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Configurando a porta do servidor
server_port = 1111 # Porta do servidor

# Configurando o endereço do servidor
server_address = ('', server_port) # Endereço do servidor (v

# Vinculando o socket do servidor ao endereço e porta
server_sock.bind(server_address)

# Esperando por uma mensagem do cliente
message, client_address = server_sock.recvfrom(1024) # Taman

# Convertendo a porta do cliente de volta para a ordem de byte
client_port = socket.ntohs(client_address[1])

# Exibindo a porta do cliente
print(f"Porta do cliente: {client_port}")

# Fechando o socket do servidor
server_sock.close()
```

Lendo um Arquivo de Texto:

- Para abrir e ler um arquivo de texto em Python, usamos a função `open()` e o método `read()`.
- Se o arquivo não for encontrado, capturamos a exceção `IOError`.
- Aqui está um exemplo:

```
filename = 'foo.txt'
try:
    inputfile = open(filename, 'r')
except IOError:
    print('Arquivo não encontrado')
contents = inputfile.read()
print(contents)
```

Dividindo Linhas de uma Mensagem:

- Quando recebemos dados de um soquete, recebemos um grande bloco de bytes.
- Precisamos converter esses bytes em uma string e, em seguida, dividi-los em linhas.

```
message = 'hello there'
parts = message.split(' ')
print('Primeira parte: ' + parts[0])
print('Segunda parte: ' + parts[1])
```

Além disso, algumas bibliotecas úteis para o projeto incluem `socket` (essencial), `time` ou `datetime` (para datas), `re` (expressões regulares), `os.path` e/ou `os.stat` (para informações de arquivo). Certifique-se de obter aprovação para usar outras bibliotecas ou utilitários além desses.