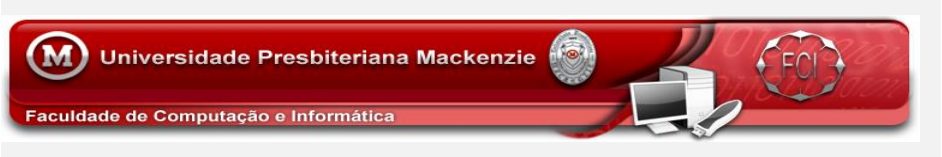


Redes de computares

Prof. Dr. Bruno da Silva Rodrigues
Bruno.rodrigues@mackenzie.br

Análise Programação de Socket UDP e TCP

Procedimento

- *Execute a IDLE do Python e abra os arquivos (Cliente e servidor) do socket UDP*
- *Cada arquivo deve ser aberto em uma IDLE diferente ou em computadores diferentes.*
- *Faça o mesmo procedimento para os arquivos socket TCP*

Diego Souza Lima Marques – TIA: 32039921

Após abrir o arquivo analise os pacotes e responda:

Questão 1*. Compile dos programas (cliente e servidor) TCP e UDP:*

a) *Execute o cliente TCP antes de executar o servidor TCP. O que acontece? Por quê?*

R: Ao executar o cliente TCP antes de executar o servidor, acontece um erro do tipo ConnectionRefusedError, em que a conexão não pôde ser feita devido ao fato da máquina de destino recusar qualquer conexão ativamente. Isso acontece porque, se o servidor não está disponível, a conexão TCP não pode ser estabelecida. Por outro lado, caso o servidor TCP estivesse sido inicializado primeiramente, o servidor estaria aguardando mensagens e qualquer tentativa de iniciação de uma conexão – quando detectasse alguma, iria tentar iniciar a conexão baseado no IP e na porta especificados.

Objetivos da atividade:

-Aprender a programar e Socket UDP e TCP assim como analisar e comparar o funcionamento de ambos protocolos da camada de transporte

Bibliografias

KUROSE, J. F. e ROSS, K. W. Redes de Computadores e a Internet – Uma Nova Abordagem – Pearson

Deitel, Harvey M.; Deitel, Paul J. - Java: como programar - 6ª edição - Pearson

Internet Engineering Task Force. Disponível em: https://www.ietf.org/t

b) *Faça o mesmo procedimento para o cliente e servidor UDP. O resultado foi similar ao socket TCP? Compare os resultados e justifique.*

R: O protocolo UDP é conhecido por ser menos seguro e confiável, isto é, no sentido de garantir a entrega íntegra de pacotes. Logo, ao executar o cliente UDP antes do servidor UDP, o resultado é diferente: a mensagem do cliente é enviada e a execução termina normalmente. Em outras palavras, o protocolo UDP garante que a mensagem vai ser enviada de qualquer maneira, mas não garante que ela vai chegar, e não se importa caso pacotes foram perdidos, ou caso o servidor não tenha a recebido. Com esse procedimento, o cliente simplesmente envia a mensagem, mas não garante que chegou, que deu problema ou que o servidor a recebeu, e é por esse motivo que não dá erro como deu no TCP, porque o protocolo simplesmente não se importa com a integridade da transferência de pacotes.

c) *O que acontece se o número da porta que o cliente tentar se conectar for diferente da porta disponibilizada pelo servidor?*

R: No caso do protocolo UDP, a mensagem do cliente continua sendo enviada, não importando se a porta está certa, ou se o servidor recebeu a mensagem. Porém, o servidor não recebe a mensagem, visto que a porta especificada foi outra.

Já para o protocolo TCP, novamente, por verificar e garantir a integridade e recebimento de pacotes/mensagens, quando o cliente utiliza uma porta diferente da especificada pelo servidor, acontece um erro de ConnectionRefusedError para o cliente. Isso ocorre porque a porta correta não foi utilizada e a conexão não foi estabelecida, além de que o servidor não vai conseguir reconhecer as informações em uma porta diferente da qual está instaurado.

Questão 2. *Faça um chat entre cliente servidor (UDP ou TCP) onde ambos os lados trocam mensagens até uma das partes enviar o comando QUIT.* ***A porta do socket deve ser os primeiros 5 números do seu TIA.***

Observações: não sei se foi só algum erro de iniciante, mas não consegui realizar o procedimento por meio do protocolo UDP utilizando uma mesma máquina. Logo, utilizei o protocolo TCP para a atividade. Outra coisa que pesquisei foi o comando decode, visto que a mensagem "QUIT" não estava sendo interpretada corretamente.

Código do servidor TCP:

ServerTCP.py - C:\Users\diego\Documents\Ciência da Computação\5º PERÍODO\REDES DE COMPUTADORES - 05D\- REDES DE

File Edit Format Run Options Window Help

```
import socket #importa modulo socket

TCP_IP = '192.168.0.103' # endereço IP do servidor
TCP_PORTA = 32039        # porta disponibilizada pelo servidor
TAMANHO_BUFFER = 1024      # definição do tamanho do buffer

# Criação de socket TCP
# SOCK_STREAM, indica que será TCP.
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# IP e porta que o servidor deve aguardar a conexão
servidor.bind((TCP_IP, TCP_PORTA))

#Define o limite de conexões.
servidor.listen()

# Aceita conexão
conn, addr = servidor.accept()

while True:
    #dados retidados da mensagem recebida
    data = conn.recv(TAMANHO_BUFFER).decode('UTF-8')
    if data == 'QUIT':
        break
    else:
        print ("Mensagem recebida do cliente: ", data)
        MENSAGEM = input("Digite uma mensagem para o cliente: ")
        if MENSAGEM == 'QUIT':
            conn.send(MENSAGEM.encode('UTF-8'))
            break
        print("Mensagem enviada para o cliente: ", MENSAGEM)
        conn.send(MENSAGEM.encode('UTF-8'))

conn.close()
servidor.close()
```

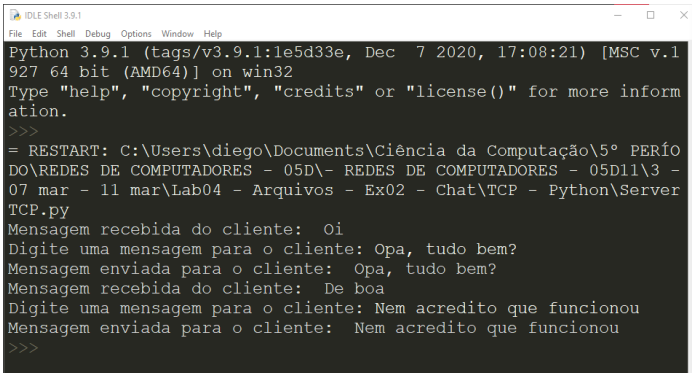
Código do cliente TCP:

ClientTCP.py - C:\Users\diego\Documents\Ciência da Computação\5º PERÍODO\REDES DE COMPUTADORES - 05D\- REDES DE COMPUTADORE

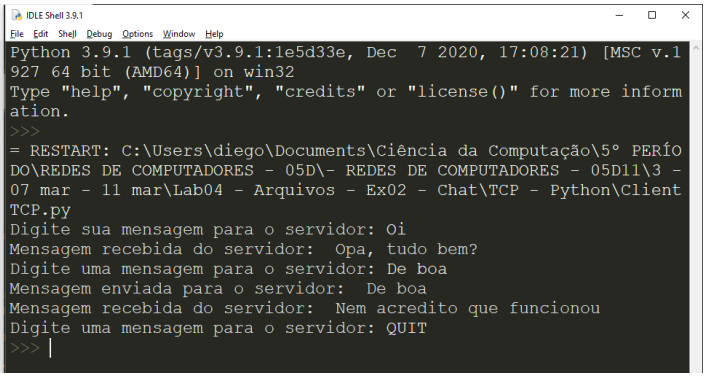
File Edit Format Run Options Window Help

```
import socket #importa modulo socket

TCP_IP = '192.168.0.103' # endereço IP do servidor
TCP_PORTA = 32039       # porta disponibilizada pelo servidor
TAMANHO_BUFFER = 1024

MENSAGEM  = input("Digite sua mensagem para o servidor: ")

# Criação de socket TCP do cliente
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Conecta ao servidor em IP e porta especifica
cliente.connect((TCP_IP, TCP_PORTA))

# envia mensagem para servidor
cliente.send(MENSAGEM.encode('UTF-8'))

while True:
    data = cliente.recv(1024).decode('UTF-8')
    if data == 'QUIT':
        break
    else:
        print("Mensagem recebida do servidor: ", data)
        MENSAGEM = input("Digite uma mensagem para o servidor: ")
        if MENSAGEM == 'QUIT':
            cliente.send(MENSAGEM.encode('UTF-8'))
            break
        print("Mensagem enviada para o servidor: ", MENSAGEM)
        cliente.send(MENSAGEM.encode('UTF-8'))

# fecha conexão com servidor
cliente.close()
```

Teste de execução:



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1
927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inform
ation.
>>>
= RESTART: C:\Users\diego\Documents\Ciência da Computação\5º PERÍO
DO\REDES DE COMPUTADORES - 05D\- REDES DE COMPUTADORES - 05D11\3 -
07 mar - 11 mar\Lab04 - Arquivos - Ex02 - Chat\TCP - Python\Server
TCP.py
Mensagem recebida do cliente:  Oi
Digite uma mensagem para o cliente: Opa, tudo bem?
Mensagem enviada para o cliente:  Opa, tudo bem?
Mensagem recebida do cliente:  De boa
Digite uma mensagem para o cliente: Nem acredito que funcionou
Mensagem enviada para o cliente:  Nem acredito que funcionou
>>>
```



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1
927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inform
ation.
>>>
= RESTART: C:\Users\diego\Documents\Ciência da Computação\5º PERÍO
DO\REDES DE COMPUTADORES - 05D\- REDES DE COMPUTADORES - 05D11\3 -
07 mar - 11 mar\Lab04 - Arquivos - Ex02 - Chat\TCP - Python\Client
TCP.py
Digite sua mensagem para o servidor: Oi
Mensagem recebida do servidor:  Opa, tudo bem?
Digite uma mensagem para o servidor: De boa
Mensagem enviada para o servidor:  De boa
Mensagem recebida do servidor:  Nem acredito que funcionou
Digite uma mensagem para o servidor: QUIT
>>>
```

(Servidor) **(Cliente)**

Enviarei os arquivos de códigos zipados em anexo para verificação.

Questão 3. *Faça uma aplicação qualquer usando o socket. Essa aplicação pode ser para enviar arquivos ou controlar algum objeto em uma das pontas da conexão ou gerenciar diversas conexões usado threads. Após elaboração do projeto, um vídeo deve ser gravado mostrando o funcionamento da aplicação e explicando o código.*

Tratar arquivos

Se a opção escolhida for manipular arquivos, um dos programas deverá dar ao usuário a opção de escolher qual arquivo será enviado e transmitir o arquivo via socket.

****será necessário estudar manipulação de arquivos. Não serão aceitos projetos onde o arquivo a ser enviado já é predefinido no código.***

Tratar várias conexões

Neste desafio, o programa será capaz de tratar e responder diversas conexões usando threads

**será necessário estudar threads.*

Outras interações

Você pode usar seu projeto ou conceitos de ***Pygame*** *usado em semestres anteriores e controlar objetos usando um computador remoto através do socket.*

A opção escolhida foi realizar a manipulação de arquivos, na qual o cliente vai dizer o nome de um arquivo por meio do usuário, ler o conteúdo desse arquivo e mandar cada conteúdo lido para o servidor. Por sua vez, o servidor vai pegar esse conteúdo e colocar até o final em um arquivo que está criando a partir das informações recebidas. Após a criação do arquivo, o servidor vai printar o conteúdo desse arquivo .txt

Irei anexar os códigos desse exercício também.

Fontes consultadas:

Material sobre arquivos em Python da disciplina de Algoritmos e Programação II (2º semestre)

Livro-texto da disciplina: Redes de computadores e a internet: uma abordagem top-down (6ª edição)

https://pt.stackoverflow.com/questions/219968/enviando-arquivos-atrav%C3%A9s-de-sockets-python

Link para o vídeo explicando o projeto: https://youtu.be/dRyM_nivmDY

Código do servidor TCP:

ServerTCP_ex03.py - C:\Users\diego\Documents\Ciência da Computação\5º PERÍODO\REDES DE COMPUTADORES - 05D\- REDES DE COMPUTADOR...

File Edit Format Run Options Window Help

```
# Diego Souza Lima Marques - TIA: 32039921
# Lab - Socket - Desafio - Ex. 03
# Servidor TCP

import socket #importa modulo socket

TCP_IP = '192.168.0.103' # endereço IP do servidor
TCP_PORTA = 32039        # porta disponibilizada pelo servidor
TAMANHO_BUFFER = 1024     # definição do tamanho do buffer

# Criação de socket TCP
# SOCK_STREAM, indica que será TCP.
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# IP e porta que o servidor deve aguardar a conexão
servidor.bind((TCP_IP, TCP_PORTA))

#Define o limite de conexões.
servidor.listen(1)

flag_escrita = False
final_arquivo = False

while True:
    # Aceita conexão a cada 1024 bytes lidos
    conn, addr = servidor.accept()
    nome_arquivo = conn.recv(1024).decode('utf-8')
    arq_recebido = open(nome_arquivo, 'wb')
    while flag_escrita == False:
        parte_arq = conn.recv(1024) # recebe cada parte de 1024 bytes do arquivo
        while parte_arq:
            arq_recebido.write(parte_arq) # escreve no arquivo
            parte_arq = conn.recv(1024) # lê a próxima parte

        flag_escrita = True

    arq_recebido.close()
    break
```

```
print("Arquivo recebido com sucesso! Imprimindo seu conteúdo:\n")

arq_final = open(nome_arquivo, 'r', encoding='utf-8')
while not final_arquivo:
    linha = arq_final.readline()
    if linha == '': # string vazia
        final_arquivo = True
    else:
        conteudo = linha.rstrip() # tirar o caractere de controle
        print(conteudo)

arq_final.close()

conn.close()
servidor.close()
```

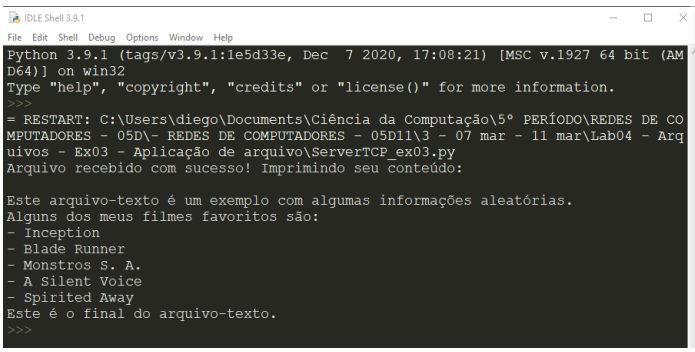
Código do cliente TCP:

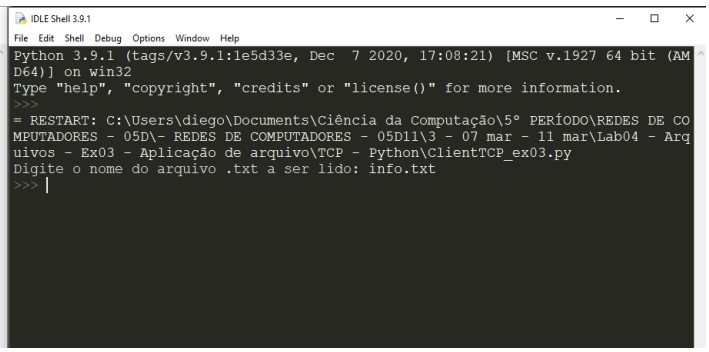
ClientTCP_ex03.py - C:\Users\diego\Documents\Ciência da Computação\5º PERÍODO\REDES DE COMPUTADORES - 05D\- REDES DE COMPUTADOR...

File Edit Format Run Options Window Help

```
# Diego Souza Lima Marques - TIA: 32039921
# Lab - Socket - Desafio - Ex. 03
# Cliente TCP

import socket #importa modulo socket

TCP_IP = '192.168.0.103' # endereço IP do servidor
TCP_PORTA = 32039        # porta disponibilizada pelo servidor

# Criação de socket TCP do cliente
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Conecta ao servidor em IP e porta especifica
cliente.connect((TCP_IP, TCP_PORTA))

nome_arquivo = input("Digite o nome do arquivo .txt a ser lido: ")

cliente.send(nome_arquivo.encode('UTF-8'))

arq = open(nome_arquivo, 'rb') # leitura de arquivo binário

parte_arq = arq.read(1024) # lê 1024 bytes do arquivo

while parte_arq: # enquanto ainda houver conteúdo para ler
    cliente.send(parte_arq) # envia o conteúdo do cliente para o servidor
    parte_arq = arq.read(1024) # lê os próximos 1024 bytes do arquivo

arq.close()

# fecha conexão com servidor
cliente.close()
```

Execução:





Notas

Questões 1 e 2 da atividade (CHAT simples) – 4,5 pontos

Questão 3 Desafio – 4,5 pontos

Vídeo com explicação do projeto – 1,0 ponto