UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO

Trabalho de redes de computadores 2021.2

Maceió/AL
2022

UNIVERSIDADE FEDERAL DE ALAGOAS

Disciplina: Redes de Computadores,
do segundo período do curso de Ciência da Computação
Professor responsável pela disciplina: Leandro Melo de Sales
Alunos responsáveis para fazer o trabalho de implementação: Daniel José da Silva e Gledson Lima dos Santos

2022

SUMÁRIO

Tema: Implentação de um chat usando Socket
Palavras-Chaves: Resdes, Socket, programacão e chat.

1 Introdução

Nesse resumo, será explicado como desenvolvemos nosso chat usando socket, funções primitivas e Thread. O projeto feito foi utilizado alguns conhecimentos da disciplina de redes de computadores e projeto inteiro foi desenvolvido com a linguagem de programação Python 3. Objetivo do projeto foi usamos conhecimentos aprendidos na disciplina e podemos aplicar não apenas nesse projeto que será explicado, mas também, como poderemos utilizar em outros projetos e tarefas futuras.

2 Resumo Código do Servidor.py e suas funcionalidades:

O socket tipo servidor que criamos é capaz de receber a conexão de vários clientes, podendo definir um limite máximo de clientes conectados. Esse servidor aceita a conexão do cliente, tornando possível a troca de mensagens entre os clientes conectados à rede. Qualquer cliente pode enviar uma mensagem para um cliente específico, ou pode enviar uma mensagem para todos os clientes conectados à rede.

O código se inicia na função **"def main()"**, inicialmente criamos o nosso socket, logo depois temos uma condicional, dentro dessa condicional, através da função bind() nós associamos(vinculamos) nosso servidor a um endereço e porta específicos, definidos previamente no código. Se alguma exceção ou erro acontecer, o servidor irá printar uma mensagem de erro.

Também temos a função **"listen()"** ela fica responsável por indicar a quantidade de conexões que serão enfileiradas pelo protocolo TCP até que o servidor realize o próximo passo, ou seja o comando accept.

Dentro do while o servidor fica aguardando a conexão do cliente, imprimindo uma mensagem quando o mesmo for conectado. Nesse momento temos a função Thread que irá executar a função **"manipular_cliente()"** em paralelo. A nossa função **"manipular_cliente()"** fica responsável por adicionar o nome, e o socket tipo cliente à nossa lista de clientes ativos.

Logo após isso temos as funções de enviar mensagem. A nossa função **"prox()"** foi criada para **"escutar"** a mensagem do cliente, e enviar para todos os clientes conectados ao servidor. Enquanto que a função **"enviar_mensagem()"** "escuta

as mensagens e enviar para um cliente específico.

3 Resumo Código do Cliente.py e suas funcionalidades:

O código começa a rodar pela nossa função **"main()"**, a primeira coisa definida foi o endereço de IP da conexão, um endereço fixo de conexão ponto a ponto. No nosso caso, como é um Socket de conexão local e fixa, rodando essa aplicação no nosso próprio computador, o endereço do nosso HOST será o IP do computador.

Para o Cliente TCP se conectar ao servidor, ele usa a função **"connect()"**, que recebe como parâmetro o endereço do servidor. Logo após isso chamamos na função **"conexao_servidor()"** que irá enviar o nome do usuário para o servidor, para que ele adicione o nome à lista de clientes conectados, verificando se o usuário digitou algo, ou deixou vazio.

Logo após partimos para a função **"enviar_msg_servidor(cliente)"** que permite o envio de mensagens para todos os clientes conectados ao servidor. Para enviar a mensagem

codificada para o servidor, nós usamos a função **"sendall()"**, enviando assim que o usuário digita a mensagem.

Assim como o servidor precisa **"escutar"** as mensagens do cliente, o cliente também precisa **"escutar"** as mensagens do servidor, para isso criamos a função **"msg _servidor(cliente)"**, onde recebemos a mensagem vinda do servidor, e verificamos se está vazia. Se a mensagem não for vazia nós recebemos o nome e mensagem do usuário, na posição 0 e posição 1 respectivamente, ou seja, essa função garante que sempre que sempre que o servidor enviar uma mensagem, ela seja exibida para o cliente.

3.1 O que poderia ter sido implementado:

Uma das coisas que poderiam ter sido implementadas é a interface gráfica, isso mesmo, essa interface teria nossa aplicação mais funcional e fácil de ser manuseada pelo usuário.

Outro quesito que poderia ter sido implementado, é fazer com que o nosso chat funcionasse em dois computadores diferentes, onde poderia haver clientes com diferentes IP's. A aplicação que construímos funciona apenas com diferentes terminais, porém na mesma máquina (computador), ou seja, nós rodamos o servidor em uma aba do terminal, depois abrimos outras para rodar o código do ou dos clientes.

3.2 Dificuldades na construção do código:

Uma das dificuldades foi fazer com que o código do servidor **"ouvisse"** a todo momento o que o cliente está escrevendo, e vice-versa. Essa função **"ouve"**, ou seja, ela está a todo momento **"aberta"** (não sei se essa seria a palavra certa), para receber as mensagens do cliente, e o cliente receber as mensagens do servidor, é como se o cliente enviasse a mensagem, o servidor recebesse e enviasse para todos que estão conectados na rede, ou para um cliente em específico.

REFERÊNCIAS

[1]**Curso de Python:** Chat em Rede com Python, https://www.youtube.com/watch?v=QYfwZ5zkGJ0list=PL84Jm3ZcNx-La$_U E_P Zl94y89au3GthY - index = 1t = 238s

[2]**Cliente e Servidor UDP:** http://www.ic.uff.br/ debora/praticas/aplicacao/: :text=A%20fun%C3%A7%C3%A3o%20listen()%20coloca,define%20o%20modo%20servidor%20s

[3]**Socket em Python:**https://blog.4linux.com.br/socket-em-python/: :text=Sockets%20s%C3%A3o%20usados%20para%20enviar,at%C3%A9%20mesmo%20as%20pr%C3%B3prias%20mensagens

[4]**Introdução a Sockets em Python:**https://medium.com/@urapython.community/introdu%C3%A7%C3%A3o-a-sockets-em-python-44d3d55c60d0

[5]**Low-level networking interfac:**https://docs.python.org/3/library/socket