***Construção de uma aplicação - Modelo IRC chat***

116572 Redes de Computadores

Profa. Priscila Solís Barreto

Autores:

Arthur

Departamento de Ciência da Computação

Universidade de Brasília

Brasília - DF, Brasil

line 4-e-mail address if desired

Enzo Zanetti Celentano (211026495)

Departamento de Ciência da Computação

Universidade de Brasília

Brasília - DF, Brasil

enzozc3@gmail.com

Paulo Fernando Vilarim de Almeida (211043351)

Departamento de Ciência da Computação

Universidade de Brasília

Brasília - DF, Brasil

paulofvilarimda@gmail.com

*Resumo – O presente artigo é um relatório elaborado a partir de um projeto de disciplina, pelo Departamento de Ciência da Computação, na disciplina de Redes de Computadores. Seu desenvolvimento baseou-se em elaborar uma aplicação capaz de implementar um servidor de bate-papo Internet, utilizando um subconjunto do protocolo Internet Relay Chat (IRC).*

*Palavras-chave - Internet, IRC, bate-papo, aplicação*

*Abstract - This article is a report developed from a discipline project, by the Computer Science Department, in the Computer Networks discipline. Its development was based on the elaboration of an application capable of implementing an Internet chat server, using a subset of the Internet Relay Chat (IRC) protocol.*

*Keywords – Internet, IRC, chat, application*

# Introdução

Aplicações do modelo cliente servidor estão presentes nas mais variadas esferas do nosso dia a dia, no que tange ao uso de aplicações de Internet. Neste contexto, um exemplo seria o Internet Relay Chat (IRC), um sistema de bate-papo em texto, constituído de clientes operando como usuários da aplicação, canais de comunicação para a efetiva troca de mensagens entre os usuários, e um servidor capaz de suportar e manipular tais operações.

Sob tal perspectiva, o presente artigo descreve a elaboração de uma aplicação baseada no protocolo do IRC, e está dividido em: fundamentação teórica, onde serão abordados os conceitos necessários para entendimento do trabalho realizado; análise do ambiente experimental e dos resultados, onde serão abordadas as descrições de hardware e software utilizadas, bem como o exame das atividades executadas no desenvolvimento da aplicação; por fim, as conclusões elucidadas a partir do artigo.

# Fundamentação Teórica

## Conceitos teóricos

A priori, a camada de aplicação é uma abstração da seção de protocolos de comunicação responsáveis pela troca de mensagens entre as aplicações. Eles atuam junto com os protocolos da camada de transporte e são responsáveis pela comuicação final entre as aplicações dos dispositivos.

A posteriori, o IRC é um protocolo que atua na camada de aplicação sob o modelo cliente servidor, ou seja, ele estabelece um bate-papo para um cliente, quando este se conecta a um servidor, com capacidade para qualquer número de clientes, o qual começa a escutá-los e a repassar suas mensagens para os demais usuários da aplicação. Outrossim, o servidor também deve ser capaz de interpretar requisições de comandos dos clientes, como a troca de seu apelido, listagem e locomoção entre os canais disponíveis, etc. As operações são feitas sob o protocolo de transporte TCP, devido à transferência de dados fim-a-fim ser realizada de forma ordenada, bidirecional e consideravelmente confiável, onde os pacotes transmitidos possuem um bit de reconhecimento, conhecido como Ack.

O IRC foi criado em 1998 e já passou por diversas mudanças desde sua primeira publicação, existindo mais de uma Request for Comments (RFC) para descrevê-lo. O IRC moderno possui uma infinidade de recursos, tais quais a presença de bots, para facilitar algumas operações do servidor, e a detecção de proxy, para arbitrar a conexão com um usuário detentor de um servidor proxy inseguro. Todavia, alguns padrões entre as implementações do IRC ainda se seguem, e dentre elas: a utilização da porta TCP 6667 ou semelhantes; a presença de canais de comunicação; a possibilidade de o usuário registrar um apelido no bate-papo; a possibilidade de o cliente enviar uma mensagem privada para um usuário específico, entre outras funcionalidades.

Ademais, pelo lado do cliente, tal usuário pode ser executado nos mais variados tipos de máquinas, tanto com diferentes hardwares, como um desktop ou um smartphone, quanto com diferentes softwares, como sistemas operacionais Windows ou Linux. Sob esse viés, o participante do bate-papo IRC estará presente na aplicação com certas atribuições, como um identificador – podendo ser uma máscara de host -, um apelido, um nome de usuário e o nome de seu host - o qual é recomendado ser borrado, para devida segurança e proteção dos clientes.

Ao se conectar com o servidor, o cliente IRC poderá lhe enviar pacotes de mensagens, enquanto ainda escuta e recebe outros pacotes do servidor IRC. Os pacotes que chegam no servidor serão examinados, devidamente manipulados e, então, serão entregues para os respectivos destinatários das mensagens, podendo ser transmitidas em um canal aberto ou de forma privada para um específico usuário.

## Técnicas utilizadas

Para o desenvolvimento da aplicação, foram utilizados sockets na linguagem de programação Python, o programa WIRESHARK, para validação da funcionalidade do código em conectar diferentes dispositivos na mesma aplicação e para exame dos pacotes trocados na rede, além da plataforma GitHub, para relato e documentação do código compartilhado entre os 3 autores do projeto.

# III. Ambiente Experimental e Análise de Resultados

## Descrição do Cenário

## Configurações utilizadas

Cada autor teve um ambiente de desenvolvimento diferente, com seus próprios dispositivos e respectivas configurações:

## *Arthur*

A implementação e o experimento do código foram realizados por meio de: um desktop, com sistema operacional Linux Fedora 37, atuando como servidor da aplicação; um laptop, de sistema operacional Linux, com distribuição Ubuntu 22.04, e atuando como um dos clientes da aplicação; e um celular Android 10, atuando como o segundo cliente da aplicação. Além disso, foram necessárias as instalações dos softwares Wireshark table (4.0.2), Python 3.11 (Linux) e QPython 3L (Android), a fim de examinar e rodar o IRC desenvolvido.

## *Enzo Zanetti Celentano*

A implementação e o experimento do código foram realizados por meio de um desktop com sistema operacional Windows 10, o qual executou a aplicação em consoles diferentes, sendo um servidor e dois clientes, a fim de validar as funcionalidades do código. Ademais, o software necessário de instalação foi o Python 3.11 para o Windows.

## *Paulo Fernando Vilarim de Almeida*

A implementação e o experimento do código foram realizados por meio de um notebook com dois sistemas operacionais distintos, Windows 11 e Windows Subsystem for Linux (WSL) com distribuição Ubuntu 22.04, onde cada um possui uma repartição em disco distinta, além de diferentes endereços IPv4. Outrossim, o único ajuste necessário para execução da aplicação foi a instalação do Python 3.11 em ambas as máquinas.

## Linguagem de programação e bibliotecas

A linguagem de programação utilizada foi o Python, por motivações de domínio da sintaxe da linguagem por parte dos autores do projeto. Dessa forma, as seguintes bibliotecas foram utilizadas:

## socket

Para instanciamento de sockets na rede, capazes de estabelecer a comunicação entre os dispositivos alocados e adequadamente conectados;

## json

Para troca de mensagens realizada de forma persistente, onde os pacotes eram convertidos em json na transmissão pela rede;

## threading

Para realização de múltiplas tarefas e chamadas de métodos ocorrerem em fluxos de execução diferentes, possibilitando, assim, tanto o servidor quanto o cliente escutarem e enviarem pacotes simultaneamente;

## time

Para colocar os hosts em repouso em momentos adequados, ou seja, com necessidade de espera da execução de uma tarefa pelo outro lado da rede (cliente esperando servidor, ou servidor esperando cliente), a fim de se evitar conflitos e erros no envio e recebimento dos pacotes.

## *Configurações na rede*

A fim de se adequar às necessidades do protocolo IRC, o código foi implementado no modelo cliente-servidor, com a criação de duas classes de objetos, uma para o cliente, denominada Cliente, e outra para o servidor, denominada Servidor, onde cada uma se aloca em um arquivo “.py” distinto, e basta executar a interpretação do arquivo da classe que deseja instanciar um objeto e, enfim, rodá-lo em sua máquina. Em suma, a interface do usuário é constituída pelo console, ou seja, por comandos de terminal realizados no repositório o qual armazena o código em execução.

Ademais, ao instanciar sockets do servidor e do cliente, foram utilizados os argumentos:

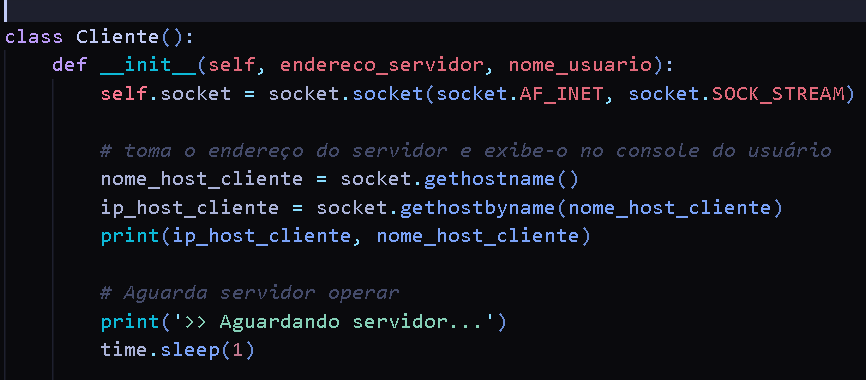
## socket.AF\_INET

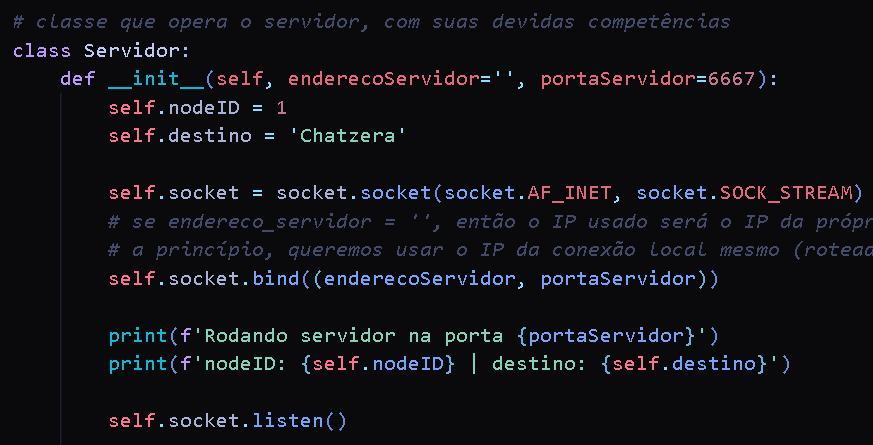
A fim de se utilizar a família de endereços IPv4, ou seja, endereços IP constituídos por quatro grupos de dois dígitos hexadecimais, variando de 0 a 255 na base decimal em exibição;

## socket.SOCK\_STREAM

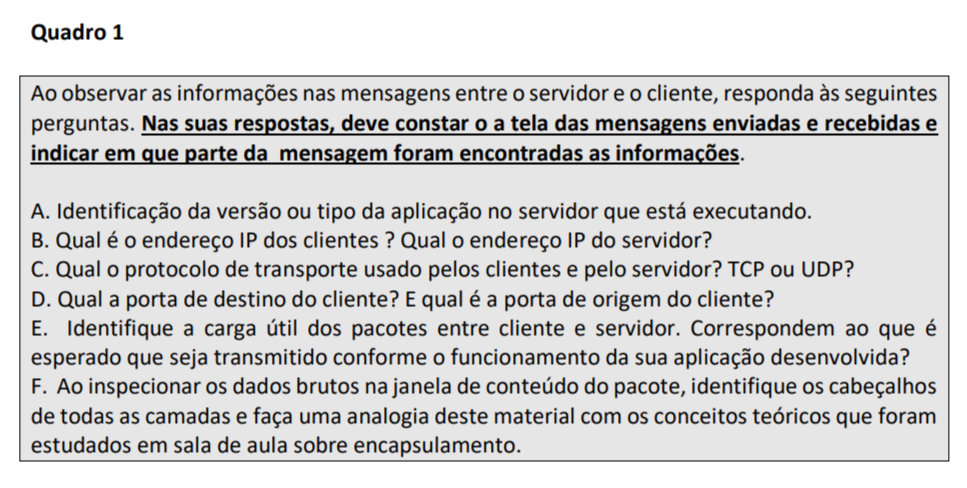
A fim de se utilizar uma transmissão de pacotes compostos por sequências de bits, sob o protocolo de transporte TCP, onde a delegação é realizada de forma ordenada, bidirecional e suficientemente confiável.

Exemplos em código:

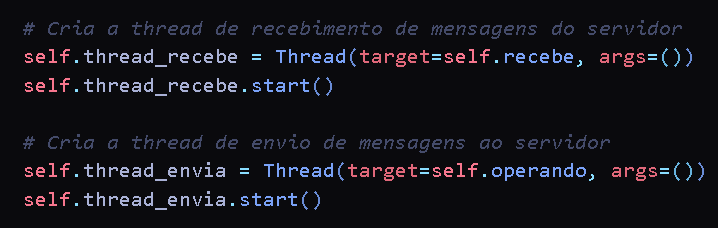
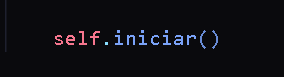
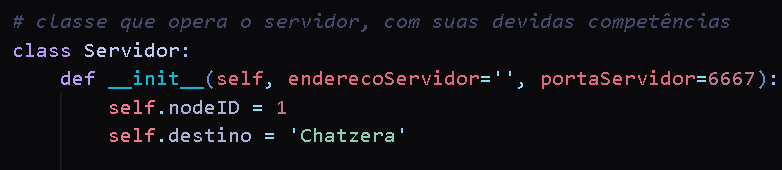




## Análise de Resultados

* A programação realizada pelos autores foi orientada a objetos e orientada a responder questões básicas impostas pelo projeto de disciplina:
* 
* A fim de responder tais indagações, foi elaborado um vídeo, por um dos autores, exemplificando a execução da aplicação desenvolvida, por meio do exame dos pacotes transmitidos na rede, com a utilização da ferramenta WIRESHARK. O link para o vídeo segue no final deste relatório, e suas análises seguem logo abaixo:

## Tipo de Aplicação

* No que tange às especificações da aplicação, ela é a implementação de um bate-papo, sob o viés do protocolo IRC, onde existe um servidor que aguarda mensagens dos clientes, para depois repassá-las aos devidos destinatários das mensagens:
* **(print mostrando o servidor recebendo e repassando mensagens aos usuários)**
* O recebimento e envio ocorre por meio de multithreading, onde, pelo lado do cliente, são criadas uma thread para recebimento de respostas do servidor e uma thread para envio de mensagens, a fim de simular um bate-papo dinâmico.
* 
* Criação das thread
* 
* Implementação das threads que são executadas
* Outrora, pelo lado do servidor, é criado um loop do tipo “while True” para constantemente aceitar a conexão de novos clientes ao servidor. Dessa forma, para cada participante conectado e, então, cadastrado no bate-papo, é criada uma thread de atendimento a esse cliente, para receber suas mensagens, processá-las de acordo com a sua leitura – identificando se é um comando ou não e, caso sendo, fazendo as devidas manipulações para atender o usuário -, a fim de, em seguida, repassar a resposta do servidor ou a própria mensagem do remetente aos destinatários.
* 
* Linha de código presente no método construtor da classe Servidor. Chama o método responsável pelas operações do servidor
* 
* Implementação do método iniciar, responsável por cadastrar o usuário no servidor, com dados recebidos por ele assim que é criada sua instância de classe, e depois é iniciada a thread de atendimento a este cliente
* Acerca das definições do servidor, ele é operado com alguns identificadores, tais quais um nodeID e um apelido de destino, implementados como atributos da classe Servidor:
* 
* Definições da classe do servidor

## Endereços IP

* Percebe-se, nas imagens que se seguem, qual o endereço ip dos respectivos servidor e clientes operados no experimento, sendo estes da família IPv4, de acordo com o instanciamento dos sockets com socket.AF\_INET:
* **(prints mostrando os endereços ip)**

## Protocolo de Transporte

* O protocolo de transporte usado em toda a aplicação é o TCP, de acordo com o instanciamento dos sockets, tanto do servidor quanto do cliente, com o argumento socket.SOCK\_STREAM, e tal fato é verificado abaixo pelo WIRESHARK:
* **(prints mostrando os protocolos de transporte)**

## Portas de Origem e de Destino

* Percebe-se que a porta de origem do cliente é aquela definida no código pela conexão do socket do cliente com o socket do servidor, ou seja, a porta 6667. Entretanto, a porta de destino é variada e depende da máquina do cliente, conforme se observa pelas portas de destino de ambos usuários registrados serem 6667, porém cada um possui uma porta de destino diferente entre si, porém constantes nas requisições que fazem:
* **(prints mostrando as portas de origem e de destino do cliente)**

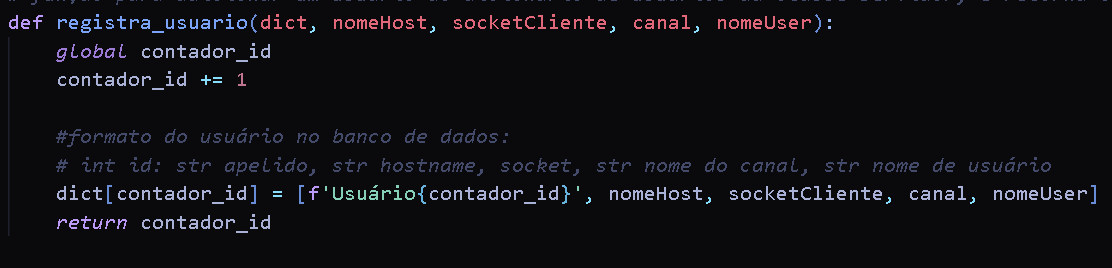
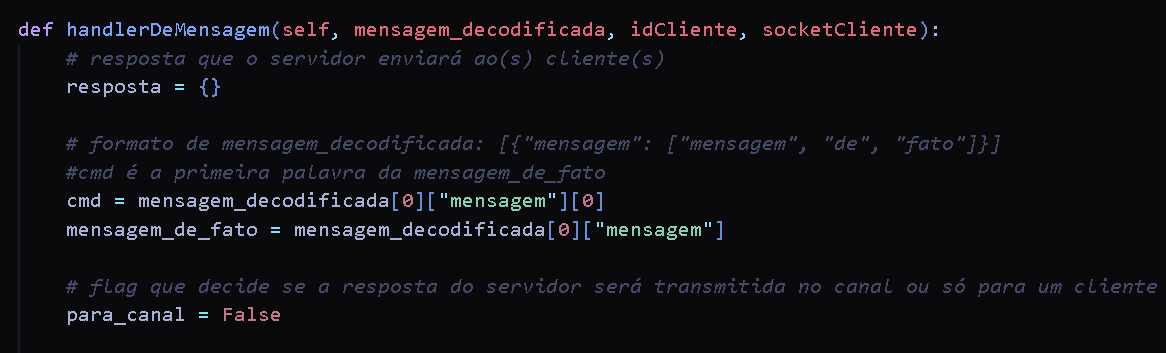
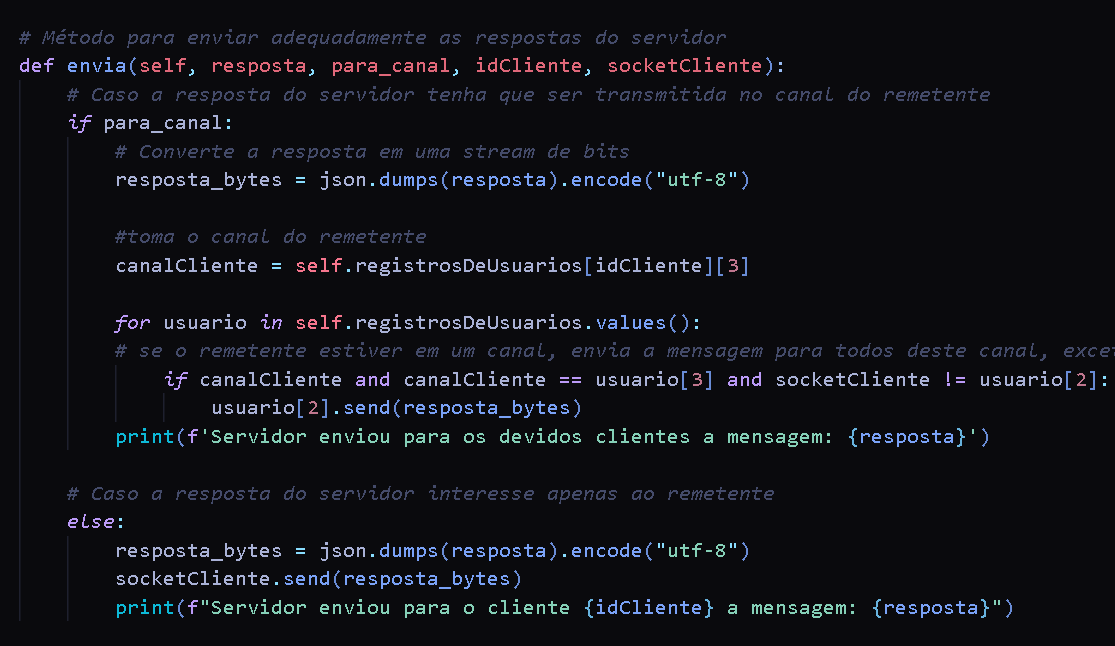
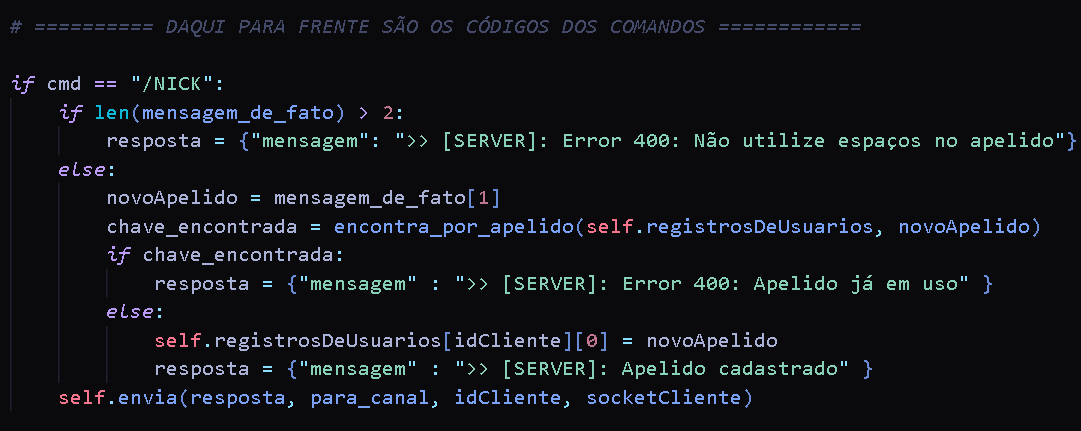
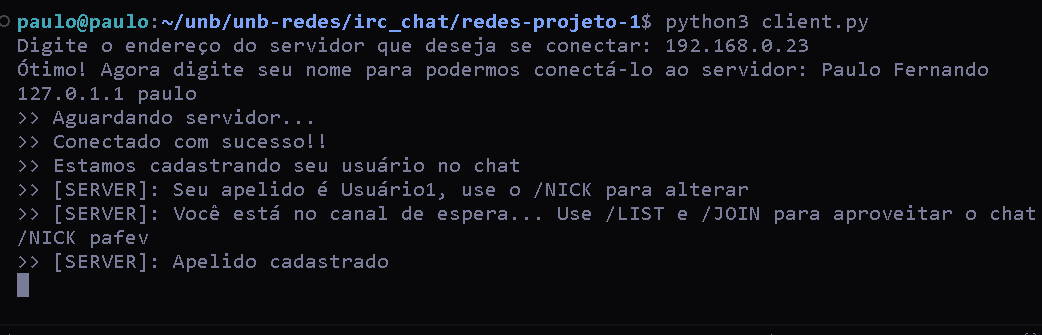
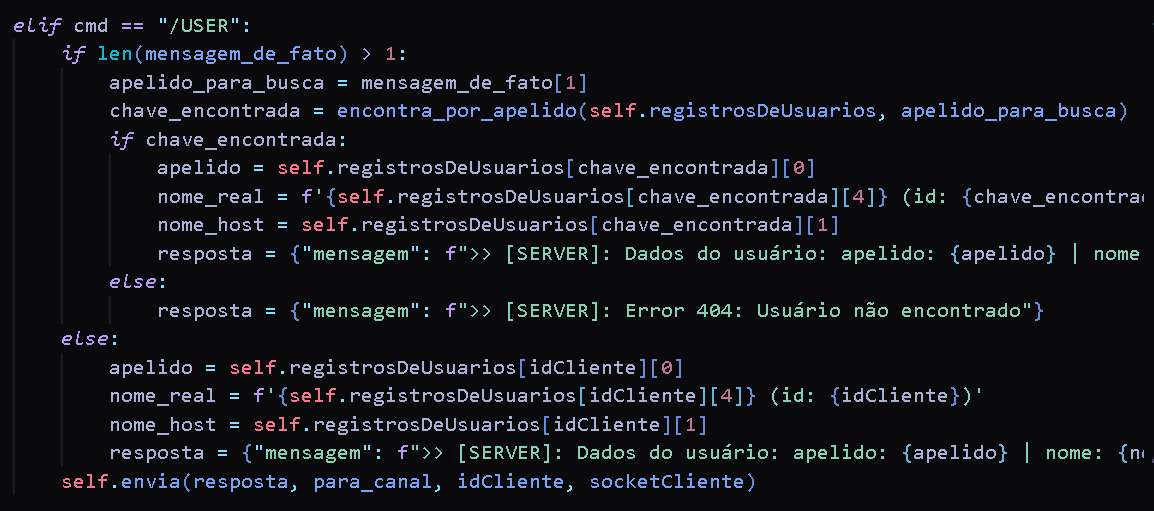
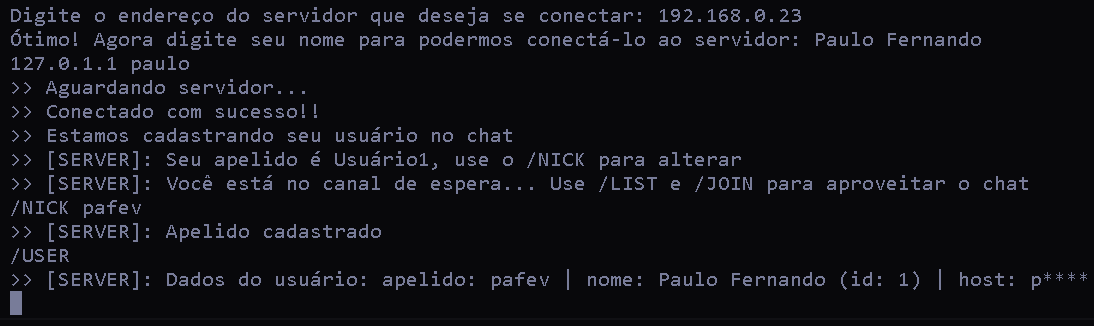
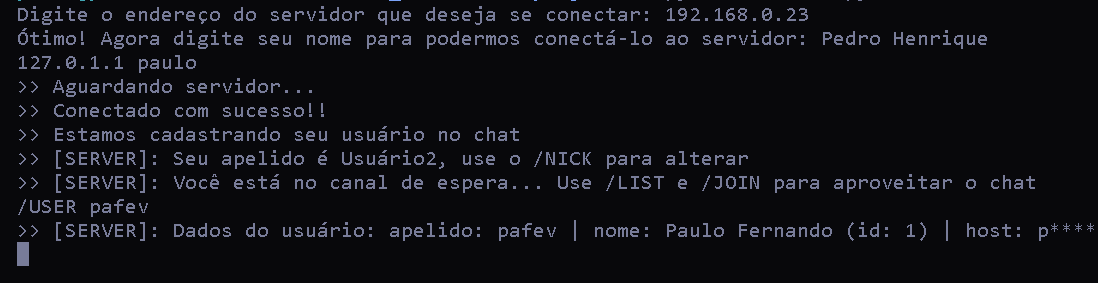
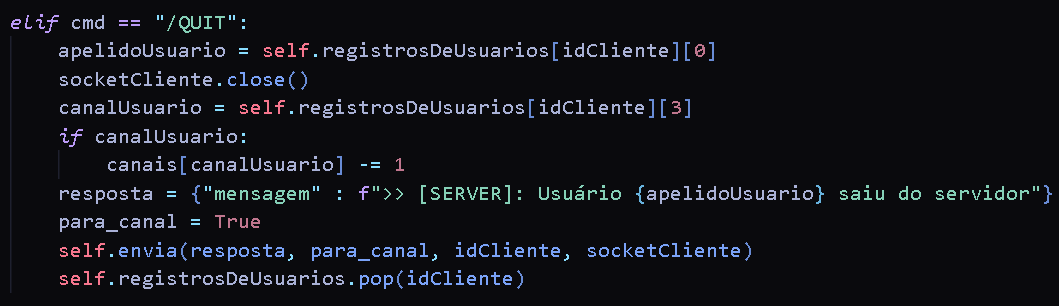
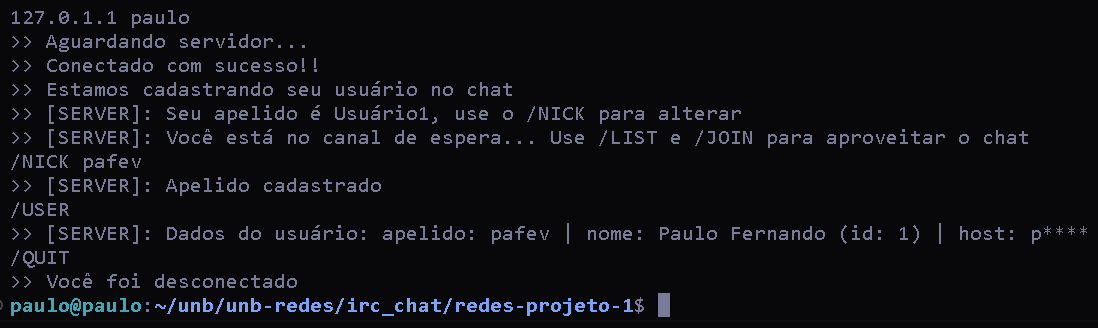
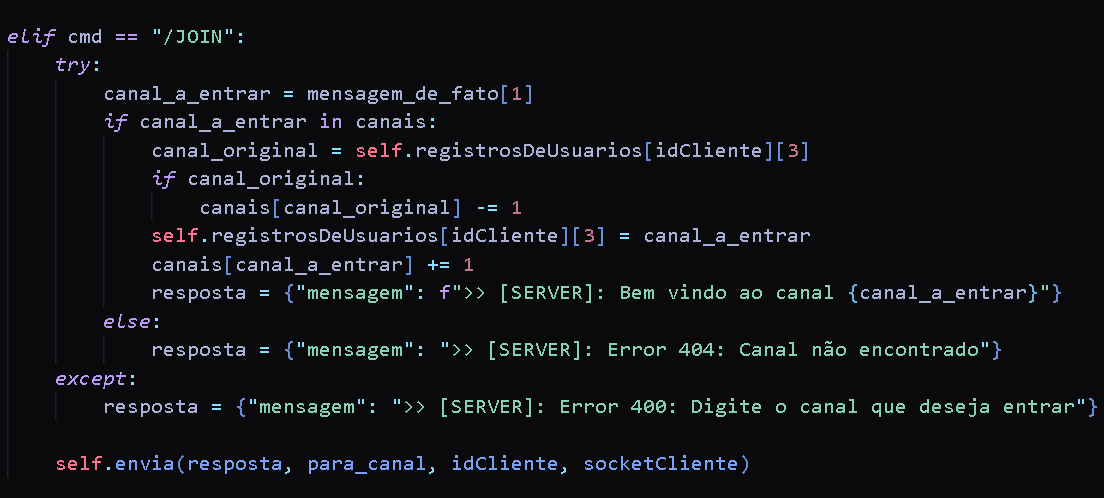
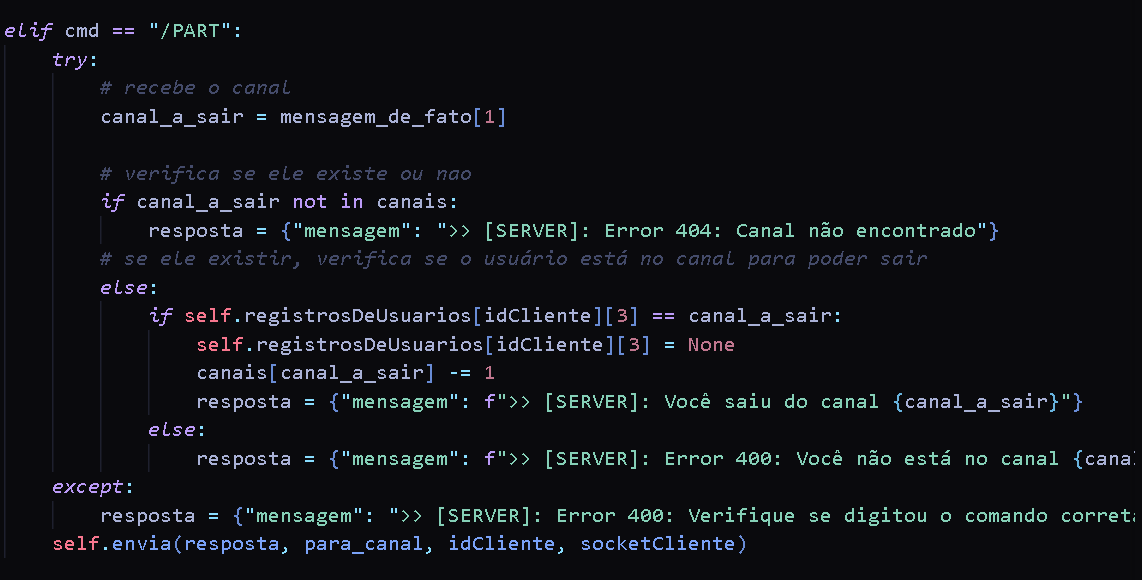
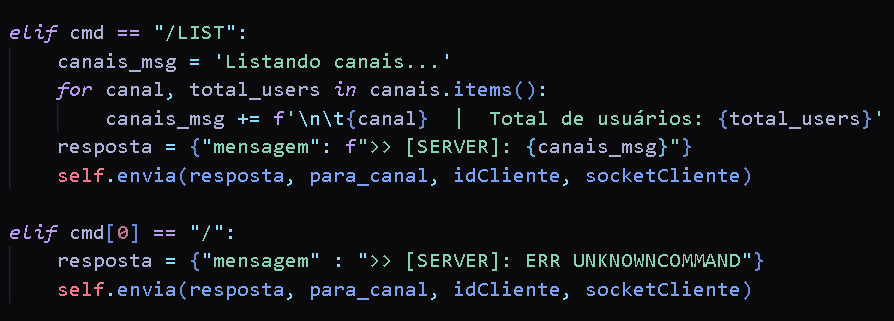
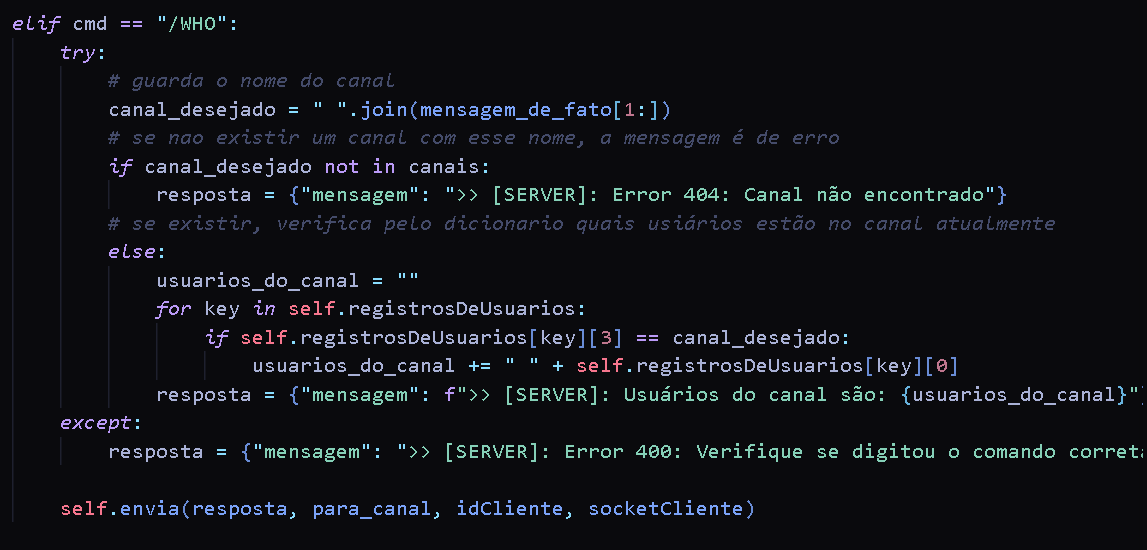
## Carga útil dos pacotes

* Na aplicação desenvolvida, foi definido um tamanho máximo de 512 bits para os pacotes recebidos pelo servidor, bem como os recebidos pelos clientes, conforme se verifica pelo código abaixo:
* 
* Nesta perspectiva, percebem-se agora os tamanhos dos pacotes examinados no WIRESHARK, onde nitidamente eles possuem um tamanho inferior àquele definido no código; e, portanto, são devidamente transmitidos e capturados na rede pelos sockets do cliente e do servidor:
* **(prints mostrando os tamanhos dos pacotes no WIRESHARK)**

## Conteúdo dos pacotes

* Percebe-se no conteúdo dos pacotes, examinados pelo WIRESHARK, a presença de quatro cabeçalhos, além do conteúdo efetivo do pacote. Nesse sentido, da imagem que se segue, as quatro primeiras linhas são os cabeçalhos: sendo o primeiro colocado pela camada física - para representar a quantidade de bits e bytes “on wire” -, o segundo, pela camada de enlace - conforme se observa a especificação do tipo de enlance utilizado; no caso, a Ethernet -, o terceiro, pela camada de redes - com a especificação da utilização do Internet Protocol Version 4 (IPv4) -, e o quarto e último cabeçalho, pela camada de transporte – com a especificação da utilização do Transmission Control Protocol (TCP). Por fim, da quinta linha em diante está contido o conteúdo efetivo da mensagem, ou “data”:
* **(print mostrando o conteúdo do pacote)**
* A ordem dos cabeçalhos faz sentido, tendo em vista que a mensagem, após ser produzida na camada de aplicação, passará em seguida pela camada de transporte, sofrendo um encapsulamento com seu respectivo cabeçalho, e, em seguida, irá para a camada de redes, depois para a camada de enlace, e, enfim, a camada física, ou seja, os “headers” são colocados e retirados em pilha no pacote.

## Comandos

* As funcionalidades desenvolvidas na aplicação foram diversas, como os comandos: /NICK, para alterar o apelido do usuário; /USER, para checar as próprias informações ou a de um usuário específico; /QUIT, para finalizar a sessão e se desconectar da aplicação; /JOIN, para se juntar a um canal em específico, ao passo que se sair do canal atual; /PART, para apenas sair de um canal específico, caso seja possível e adequado. /LIST, para listar os canais existentes no servidor local; /PRIVMSG, para redirecionar a mensagem apenas para um usuário em específico; e /WHO, para consultar os usuários de um canal específico.
* Porém, de antemão, verificou-se necessário estabelecer uma estrutura de dados capaz de armazenar as informações necessárias para execução dos códigos, como as informações dos canais e dos usuários.
* Para os usuários, foi criado um atributo dentro da classe Servidor, o qual é um dicionário capaz de armazenar os participantes do bate-papo, onde as chaves são os id’s ou identificadores únicos de cada cliente (tomados a partir do 1 e registrados pela ordem em que o cliente se conectou ao bate-papo), e os valores são listas de dados dos usuários, como apelido, nome de host, socket, canal conectado e nome real.
* 
* Função utilizada pela classe Servidor para cadastrar novos usuários no bate-papo. Após registrar, ela retorna o id desse usuário
* Para os canais, foi criado um arquivo “.py” separado, armazenando os canais oferecidos pela aplicação, e tais meios são armazenados em um dicionário, onde as chaves são os seus nomes, e os valores são as suas respectivas quantidades totais de usuários em atividade.
* 
* Assim, com as estruturas de dados utilizadas em contexto, seguem os códigos para efetiva implementação de cada comando abordado:
* Método que verifica a mensagem do cliente:
* 
* Contextualização do método para entendimento da mensagem do cliente. Os comandos abaixo fazem parte desse método
* Método responsável pelo envio da mensagem manipulada e interpretada aos devidos clientes:
* 
* /NICK
* 
* 
* 
* /USER
* 
* 
* 
* /QUIT
* 
* 
* /JOIN
* 
* /PART
* 
* /LIST
* 
* /PRIVMSG
* 
* 
* /WHO
* 

# IV. Conclusões

Em síntese, neste trabalho, percebe-se que existem diversas partes interconectadas as quais permitem o transporte de informações entre máquinas, pelo protocolo TCP. Não obstante, verifica-se também a necessidade da leitura e da compreensão de partes do protocolo para sua utilização neste projeto – especialmente, no que tange à criação e conexão dos sockets utilizados. Com isso, é crucial ressaltar que toda rede contemporânea usufrui dos assuntos abordados no trabalho, mesmo em linguagens de mais alto nível, em que as conexões são realizadas de maneira mais implícita.

##### Bibliografia

1. Site para consulta: https://pt.wikipedia.org/wiki/Internet\_Relay\_Chat
2. RFC para consulta: http://www.cs.cmu.edu/~srini/15-441/F06/project1/rfc.html
3. Exemplo de programação com socket, utilizado no começo do projeto: https://github.com/Gabrielcarvfer/Redes-de-Computadores-UnB

##### Links do código fonte e do vídeo produzidos

1. <https://github.com/akaTsunemori/redes-projeto-1>
2. (Link do vídeo)