Trabalho de Redes de Computadores

Brian Jose Costa de Medeiros
Eng. de Computação e Informação
Escola Politécnica - UFRJ
Rio de Janeiro, Brasil
brianjcmedeiros@poli.ufrj.br

Giulia Tafuri Rodrigues da Silva
Eng. de Controle e Automação
Escola Politécnica - UFRJ
Rio de Janeiro, Brasil
giutafuri.20221@poli.ufrj.br

Marcos Vinícius Theodoro P. da Silva
Eng. de Controle e Automação
Escola Politécnica - UFRJ
Rio de Janeiro, Brasil
marcos_theodoro.20221@poli.ufrj.br

Pedro Hable Miller
Eng. de Controle e Automação
Escola Politécnica - UFRJ
Rio de Janeiro, Brasil
ph19miller.20221@poli.ufrj.br

***Abstract*—Este trabalho tem como objetivo descrever o protocolo escolhido pela turma para uma aplicação pensada e desenvolvida na disciplina de Redes de Computadores. A aplicação consiste em um chat aberto para troca de mensagens entre alunos da disciplina.**

***Index Terms*—Chat, Cliente, Servidor, Socket, TCP, UTF-8**

I. Introdução

A aplicação acordada compreende um chat desenvolvido em Python que permite a troca de mensagens entre sistemas finais conectados em uma mesma rede. Com base nisso, a turma produziu e aderiu um protocolo condizente com a aplicação selecionada.

II. Aplicação

A interface escolhida pelo grupo é uma página web responsável por coletar a mensagem digitada no campo de texto e enviá-la. Além disso, a mesma opera em conjunto com um terminal que registra as mensagens enviadas por todos os usuários conectados a partir da inicialização da aplicação.

Fig. 1. Interface desenvolvida em execução.

A arquitetura aderida é cliente-servidor, com o propósito de estabelecer um sistema final dedicado (servidor) que cumpre o papel de ponte entre todos os sistemas finais que trocam mensagens entre si (clientes).

A. Servidor

Na aplicação, o servidor possui um endereço IPv4 cedido pelo DHCP na rede local, o qual é público a todos os clientes. Além disso, a porta na qual o servidor estará ouvindo as mensagens enviadas é fixo (porta 18000). Dessa forma, torna-se mais simples o processo de conexão entre cada um dos clientes e o servidor.

Utilizando o *socket* em Python, o servidor se fixa na porta e IP estabelecidos. Após isso, o mesmo cria um vetor para armazenar cada conexão realizada. Sendo assim, o servidor já está pronto para captar mensagens enviadas pelos clientes.

Para isso, o mesmo inicia uma *thread* para cada nova conexão aceita e aguarda por mensagens. Uma nova mensagem é detectada pelo envio da quantidade de caracteres da mesma juntamente com o texto escrito.

Assim que uma nova mensagem é detectada, o servidor verifica se a mesma é caracterizada como desconexão (comando ':D'). Caso afirmativo, o servidor encerra a conexão com o respectivo cliente. Caso contrário, o servidor registra a mensagem no terminal e a envia para cada um dos clientes conectados.

As mensagens trocadas pelo servidor e pelo cliente possuem comprimento máximo de 64 bytes no padrão UTF-8.

Além disso, o servidor ainda realiza algumas tarefas adicionais como registrar o número de conexões ativas, registrar cada conexão nova e desconectar devidamente cada conexão encerrada, removendo também do vetor de conexões.

B. Cliente

Na aplicação, o cliente não possui IP e porta preestabelecidos. Entretanto, deve-se garantir que ele esteja na mesma rede que o servidor com IP e porta do mesmo corretamente fornecidos.

Sabendo o endereço completo do servidor, a aplicação inicia conectando-se ao mesmo, executando a interface no navegador e abrindo uma *thread* para captar as mensagens enviadas pelo servidor. Desse modo, o terminal fica responsável por mostrar as mensagens trocadas enquanto a interface fica responsável por captar o texto inserido no campo principal.

Para desenvolver toda a interface do servidor, o grupo utilizou o Flask para renderizar páginas HTML (HyperText Markup Language) em um *localhost*.

Utilizando requisições por formulário, a interface é capaz de coletar o texto inserido pelo usuário e verifica se o mesmo é caracterizado como desconexão (comando ':D'). Caso afirmativo, o cliente fecha a sua conexão com o servidor e encerra a sua execução. Caso contrário, todo o programa continua em execução, enviando mensagens e ouvindo o servidor.

III. Transporte

Considerando a aplicação selecionada, a turma optou por rodar o chat de mensagens em cima do TCP (Transfer Control Protocol). O mesmo fornece transferência confiável de dados usando reconhecimentos positivos e temporizadores, além de garantir entregas ordenadas e sem erros.

Ademais, ainda torna-se necessário comentar sobre as características da aplicação quanto à escolha do TCP. Ele torna a aplicação orientada a conexão devido ao *three-way handshake*.

Ao mesmo tempo, a aplicação é tolerante a atrasos, visto que as mensagens podem chegar com um pequeno delay, e não opera em tempo real. Além disso, a aplicação pode ser considerada tolerante a perdas, pois apenas o TCP é responsável por tantar recuperar as mensagens perdidas na comunicação.

IV. Rede

A aplicação leva em consideração a presença de um comutador (roteador) ao qual todos os sistemas finais se conectam. Dessa forma, quando um dispositivo se conecta, o DHCP fica responsável por gerar um endereço IPv4 para o mesmo. Com as informações de IP do servidor, considerando a porta fixa (porta 18000), e múltiplos clientes conectados à rede, torna-se possível a execução da aplicação.

Sendo assim, cada dispositivo desempenha seu papel executando o programa desenvolvido e todos os clientes estão aptos a trocarem mensagens entre si.

V. Conclusão

A aplicação foi testada em sala com um *smartphone* cumprindo o papel de comutador (roteador) e no mínimo dois clientes trocando mensagens entre si. Tanto o servidor quanto a aplicação desenvolvida pelo grupo se conectaram, permitindo o envio e recebimento de mensagens até mesmo com mais de 2 clientes simultâneos sem erros de execução.