Trabalho prático 3

Redes de computadores

Aluno: Eduardo Ávila Vilar
Matrícula: 2017001796

23/12/2022

Universidade Federal
de Minas Gerais
– · –
Instituto de Ciências Exatas
Ciência da Computação

Relatório do trabalho prático 1

Terceiro Relatório da disciplina de Redes de Computadores do Departament ode Ciência apresentado no curso de Engenharia de Sistemas da Universidade Federal de Minas Gerais.

Aluno: Eduardo Ávila Vilar

Professor: Dorgival Olavo Guedes Neto

Dezembro
2022

Sumário

1 Introdução

O trabalho prático 3 da disciplina de Redes de Computadores consiste em implementar, em Python, um chat entre duas ou mais aplicações clientes onde todas as mensagens passam por um programa servidor antes. Todas as mensagens são confirmadas após serem enviadas. Clientes novos devem enviar uma mensagem "oi" e ao sair, uma mensagem "flw". Tanto o cliente quanto o servidor devem usar select para administrar múltiplas possibilidades que podem ocorrer concomitantemente de forma que uma não bloqueie a outra. No caso do servidor, ele deve estar preparado para receber mensagens por qualquer dos soquetes dos clientes, bem como enviar mensagens de volta para eles. Já os clientes devem ser capazes de enviar mensagens para outros clientes, receber mensagens do servidor e exibir outras mensagens na tela.

O programa deve ser construído sobre o TCP e funciona com base em 5 tipos de pacotes: tipo *ok*, que pode ser enviado tanto pelo servidor quanto pelo cliente e serve para confirmar que alguma mensagem foi recebida. Tipo *erro*, enviada somente pelo servidor para indicar que embora a mensagem tenha sido recebida, o seu objetivo não foi cumprido, por exemplo, quando um cliente não consegue se conectar ao servidor. Tipo *oi*, enviada pelos clientes para o servidor tão logo quando uma conexão é estabelecida entre esses dois. Funciona, portanto, como um 2-way handshake. Tipo *flw*, também enviada pelo cliente para o servidor, visando avisar este último sobre a saída do primeiro. E tipo *msg*, enviada pelos dois e tipo principal do programa. Carrega consigo uma mensagem para um cliente ao para todos eles, sendo central para a comunicação em geral no chat.

Todos os tipos têm quatro campos pelo menos, todos de dois bytes: tipo, sendo um dos citados acima. Transmissor, sendo um número que o cliente escolhe ao se conectar, ou 65535, no caso do servidor. Receptor, sendo o número do cliente o qual se destina a mensagem. Número de sequência, essencial para as confirmações de cada mensagem. Para mensagens do tipo "msg", dois campos a mais deve ser adicionados. O primeiro trata-se do tamanho da carga. O segundo é a carga em si, ou seja, o texto que deve chegar ao outro lado.

2 Arquitetura

O programa foi pensado para ser feito em três partes: `cliente.py`, `servidor.py` e `comum.py`.

O arquivo comum implementa objetos e funções que são comuns a ambas partes da aplicação, cliente e servidor. Por exemplo, um objeto `Protocol` responsável por manter o número de sequência das mensagens dos clientes, por impedi-los de usar funções destinadas somente ao servidor, e por ajudar ambos a criar pacotes no formato certo e já em bytes, para que possam ser enviados e recebidos pelo soquete. Neste arquivo ainda se encontra uma função responsável

por extrair as informações de pacote independente do seu tipo.

O cliente começa com as restrições básicas: seu identificador deve ser um número entre 1 e 65535 e ele deve enviar uma mensagem do tipo "oi" assim que se conectar com o servidor. Ele mantém 2 filas: uma para as mensagens que ele já escreveu e devem ser mandadas pelo seu soquete, outra para as mensagens recebidas que devem ser exibidas em sua tela. Na função select, três arquivos são mantidos sobre observação: o seu soquete, responsável pela comunicação com o servidor e, por consequência, com os outros cliente. Sua entrada *stdin* e sua saída *stdout*. O primeiro deve coletar as mensagens que o cliente escreve e colocá-las na fila para serem enviada através do soquete, enquanto o segundo coleta as mensagens na fila recebidas pelo soquete e as exibe na tela.

Já o servidor, começa criando o seu soquete sem host e com porta 5555 e o deixa no modo escuta, de forma que todos possam se conectar a ele. Um dicionário é utilizado visando ligar a cada chave numérica, um soquete de um cliente, sendo a primeira o seu identificador. Em seguida, ele cria duas listas e uma fila. Uma lista com todos os soquetes que ele deve observar esperando entradas e um com todos que ele deve vigiar aguardando para enviar algo por eles. A fila guarda as mensagens recebidas por ele, mas que precisam ser reencaminhadas. Dentro do select, o seu próprio soquete fica à espera de novos clientes e de seus respectivos ois. Quando esse procedimento inicial é cumprido, ele os coloca na lista de observação de entradas, já que o servidor nunca responde sem ser perguntado antes. Caso haja um erro na requisição do cliente, como o caso em que um cliente tenta se identificar com um número que já está sendo usado por outro, uma mensagem do tipo "erro" é enviada nessa fase também. As mensagens dos clientes que querem se comunicar com outros ou se desligar do sistema são recebidas no próximo passo, onde os soquetes que não são o soquete do próprio servidor são observados. Caso um cliente envie uma mensagem, esta é conferida de forma a saber para quem se deseja reencaminhar. Depois disto, ela é colocada na lista de envios para ser entregue ao cliente destino no próximo passo. Depois de observar todos os soquetes da lista de receptores, as lista de transmissores é observada. Nela, todos os soquetes que têm mensagens a serem enviadas são usados para acabar com essa pendência. Caso a mensagem não consiga passar pelo soquete, isso é tratado como um erro com o soquete e ele é desconectado do sistema. Seu soquete é retirado das listas de recepção e transmissão, além do dicionário de clientes.

3 Discussão

O programa, de forma completa, não foi concluído. Principalmente, houveram dificuldades com a função select no cliente. Ela parecia não dar prosseguimento com as observações ao *stdin* nem ao *stdout* e por isso não chegava a enviar o que devia ao soquete do servidor.

Num teste feito antes com um programa cliente mais simples, que apenas enviava sem esperar receber nada mais do que a confirmação do servidor, este último pareceu ter executado bem as funções propostas com o seu select.