REDES DE COMPUTADORES
TRABALHO PRÁTICO 2

Rafael Rubbioli : 2014124838 Fernanda Ramalho : 2014106368
Departamento de Ciência da computação, UFMG

2 de setembro 2017

1 Introdução

Este trabalho tem como objetivo a implementação de uma aplicação de troca de mensagens. Para tal, será usado um servidor e clientes TCP/IP.

2 Conceitos e ferramentas de apoio

2.1 Sockets

O programa é implementado na linguagem Python e usa sockets para fazer a comunicação. Sockets são usados para conectar clientes a uma porta e trocar mensagens. Isso ocorre em um processo chamado de abertura passiva e abertura ativa. O servidor faz a abertura passiva com o 'bind' e 'listen' e espera as conexões de clientes com 'accept'. Já os clientes fazem 'connect' no porto designado. Depois disso a troca de mensagens pode acontecer normalmente.

2.2 Select

A leitura de entrada pelo teclado e as funções de 'send()' e 'recv()' fazem com que o programa pare e espere uma resposta. Como a aplicação de chat não pode parar fazemos o uso de uma função chamada select. Essa função retorna as listas de possíveis inputs, outputs e exceções. Com isso, não é preciso esperar e deixar o programa parado, basta tratarmos todos as entradas e saídas a medida que elas acontecem.

3 Implementação

A implementação foi simples depois que o select estava funcionando. As maiores dificuldades encontradas foram para tratar a lista de IDs dos clientes. Para resolver esse problema usamos um 'dict' do python que relaciona 2 entidades, no caso o socket com o seu ID.
Outra dificuldade encontrada foi o recebimento de mensagens em network byte order. Em python, precisamos usar as funções 'struct.pack' e 'struct.unpack' para fazer essa converção, mas isso gera um 'string' que seria mais difícil de tratar, por isso usamos python3 que transforma isso em bytes, que facilita no tratamento. Isso, porém, gerou um problema para enviar o 'string' da mensagem tivemos que transformá-lo em bytes para concatenar com o restante da mensagem que havia sido transformada pelo 'struct.pack', para isso foi preciso fazer o '.encode' e o '.decode' ao enviar e receber esse string. Por fim, foi uma questão de balancear os pontos positivos e, decidimos manter o python3.
Outra dificuldade foi a implementação do timeout, já que precisávamos verificar sempre que um cliente mandava uma mensagem e retirá-lo do sistema se fosse necessário. A solução encontrada foi retirar o socket daquele cliente, então quando outros clientes pedem a lista, ele já não está mais nela.

3.1 Cabeçalho

O cabeçalho das mensagens é divido da seguinte maneira

TIPO Tipo da mensagem. Divido de 1 a 7. 2 bytes.

ORIGEM ID do cliente que enviou a mensagem. 2 bytes.

DESTINO ID do cliente que a mensagem deve ser enviada. 2 bytes.

NÚMERO DE SEQUÊNCIA Número de sequência para controle dos acknolegements enviados. 2 bytes.

3.2 Mensagens

As mensagens implementadas para o funcionamento do programa foram:

1 = OK Mensagem de acknolegement confirmando a recepção da mensagem de número de sequência do igual ao do cabeçalho.

2 = ERRO Mensagem semelhante ao OK, porém dizendo que algo deu errado na mensagem do número de sequência indicado.

3 = OI Mensagem que o cliente deve enviar ao criar conexão para receber seu identificador ID. Essa mensagem tem a origem como 0 e o destino o servidor (que tem o identificador 65535).

4 = FLW Mensagem que o cliente deve enviar para sair da conexão com o servidor. Essa mensagem espera um OK antes de permitir que o cliente seja desconectado.

5 = MENSAGEM Essa é a mensagem de chat própriamente dita. Ela pode ser feita de 2 maneiras diferentes, o destino sendo 0 significa um 'broadcast', ou seja, todos os clientes conectados receberão, ou o destino como um ID válido indica que é um 'unicast' para um certo cliente. Essa mensagem espera uma confirmação OK e tem seu cabeçalho extendido com o campo LENGTH de 2 bytes que indica quantos bytes deverão ser lidos de 'payload'.

6 = CREQ Mensagem que o cliente envia para o servidor requerendo a lista de clientes. Essa mensagem espera uma confirmação CLIST.

7 = CLIST Mensagem de resposta do servidor ao CREQ com o tamanho da lista de IDs dos clientes e a lista em si.

3.3 Servidor

O servidor tem a função de receber as mensagens e repassá-las entre os seus clientes. Para o controle disso, ele usa o número de sequência, assim como as confirmações 'OK' e 'ERRO'. Outra funcionalidade importante do servidor é controlar a lista dos clientes conectados bem quanto seus respectivos IDs. Além disso, ele controla falhas de clientes, de maneira que, caso o cliente demore mais do que 5 segundos para responder, ele desconecta aquele cliente pois acredita que ele sofreu algum erro.

3.4 Cliente

Como descrito na especificação, o cliente é capaz de receber mensagens da rede e exibi-las na tela e ler mensagens do teclado e mandá-las pela rede.
Inicialmente, o cliente se conecta com o servidor, manda um OI e espera por um OK de volta. Assim que recebe a confirmação, recebe também o seu ID próprio e mostra na tela.
Depois, o cliente tem a opção de enviar mensagem, listar todos os clientes que estão conectados no servidor ou sair do sistema. Caso deseje enviar mensagem, ele tem que indicar o ID do destino e a mensagem. As mensagens fora codificadas com o encode() para mandar e, quando recebe uma mensagem, decode().
A opção de listar os clientes apenas manda a requisição para o servidor e recebe, junto com o cabeçalho da

mensagem, uma lista com todos os IDs.
O cliente, de modo semelhante ao servidor, foi feito para que receba mensagens "ao mesmo tempo". Caso o cliente esteja fazendo uma requisição de lista e ao mesmo tempo estiver recebendo uma mensagem, ele receberá as duas mensagens com os seus respectivos cabeçalhos, nada será perdido.
Por fim, uma modificação foi feita para facilitar o desenvolvimento, quando o cliente envia um FLW, o número de sequência do programa é zerado, assim quando ele receber um OK do servidor fica mais simples de saber de qual requisição aquele OK veio, para poder fechar o socket e sair do programa.

4 Funcionalidades Extras

Foi implementada uma lista de apelidos para tornar a aplicação mais prática. Isso adicionou os seguintes tipos de mensagens:

13 OIAP Mensagem na qual o cliente envia o seu novo apelido.

15 MSGAP Mensagem de troca entre clientes comum, porém o destino é o servidor e existe um campo 'length' de dois bytes que mede o tamanho do apelido e logo em seguida o campo 'nickname' com o apelido do destino.

16 CREQAP Requisição do cliente com a lista de apelidos e seus respectivos IDs.

17 CLISTAP Resposta para a mensagem 16 com a lista de apelidos e seus respectivos IDs dos clientes.

Além das funcionalidades já explicadas, o programa também pergunta ao iniciar se o cliente quer ter um apelido. Assim ele pode mandar mensagens para outros clientes através desse apelido. Caso ele não queira, ainda assim pode solicitar a lista de apelidos e mandar mensagens usando o apelido em vez do ID.
A lista de apelidos contém apenas aqueles clientes que decidiram ter um. Então se temos 3 clientes e só o 2 não quis, a lista vai conter apenas 1 e 2 e os seus respectivos apelidos. Já a lista de ID teria todos os 3.

5 Funcionamento

Para rodar a aplicação deve ser feito da seguinte maneira:
python3 servidor.py <porto>
Já os clientes devem ser rodados assim:
python3 cliente.py <IP do server> <porto>

6 Testes

Com aplicações como essa, a maneira mais simples de testar é apenas usando. Por isso, fizemos testes manualmente com intuito de gerar erros. A corretude do programa foi verificada e não encontramos erros.
Foram feitos testes mandando mensagens para clientes que nao existem, pedidos de listas, envios broadcast, mensagens que geravam timeout. Uma coisa verificada foi que, mesmo tendo apenas um cliente e ele queira enviar uma mensagem broadcast, não retorna erro, mas pode ser verificado pedindo uma lista, uqe só existe um cliente no sistema.

7 Conclusão

Fizemos uma aplicação de mensagem e fizemos uso das ferramentas de socket já conhecidas tanto quanto as novas como o 'select'. Verificamos, também, que em python, como é uma linguagem dinâmicamente tipada, é extremamente mais simples fazer uso dos sockets, pois como vimos em C nos trabalhos anteriores era necessário fazer casts e uso de coerções que tornava muito mais complexo o uso dessas ferramentas.