Trabalho Prático 2
Transmissão de arquivos usando UDP com controle sobre TCP

Jean George Alves Evangelista - 2018047021

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

jeanalves@dcc.ufmg.br

1. Introdução

O objetivo deste trabalho foi realizar a implementação de um servidor e clientes para simulação de *upload* e armazenamento de arquivos em nuvem. Clientes enviam seus arquivos ao servidor, que por sua vez fica responsável por receber e armazená-los. Deve ser usado o protocolo TCP para transmissão de mensagens de controle e o protocolo UDP para a transferência do arquivo.

2. Implementação

A implementação foi realizada na linguagem de programação Python, na versão 3.8. Optou-se pelo desenvolvimento em Python por ser uma linguagem de mais alto nível, que fornece maneiras mais simplificadas de se trabalhar com *strings*, estruturas de dados, *threads*, etc. Além disso, utilizou-se do módulo nativo *socket*, que é muito similar ao módulo de mesmo nome disponível para C, utilizado no TP1, e portanto foi possível ter o código previamente desenvolvido como base para transmissões de mensagens utilizando TCP.
Foram desenvolvidos dois programas, divididos em dois arquivos .py: cliente e servidor. Abaixo é feita uma descrição dos passos realizados durante o desenvolvimento.

2.1 Troca de Mensagens

A primeira coisa a ser implementada foi a troca de mensagens no formato especificado utilizando TCP. Nesse passo também foram realizadas as validações necessárias nos argumentos recebidos pelo programa:

- quantidade argumentos recebidos no programa: 3 para o servidor e 4 para o cliente
- nome do arquivo no cliente: se possui no máximo 15 bytes, apenas caracteres ASCII, se contém um caractere “.” e se a quantidade de caracteres após o “.” é exatamente 3.
- validação se existe um arquivo com o nome recebido

Em seguida, no cliente e servidor foram criadas as estruturas de *socket* necessárias. No servidor, o socket foi criado utilizando AF_INET6, que o define como IPv6 e em seguida foi chamado um método que desabilita o recebimento apenas de endereços IPv4 para satisfazer o requisito de funcionar com IPv6 e IPv4. No cliente, a famÍia é definida de acordo com o endereço recebido por argumento. Em seguida, no servidor foi adicionada uma *thread* para escutar mensagens de múltiplos clientes e a lógica da transmissão de mensagens foi implementada.

A lógica segue exatamente o descrito no enunciado:

1. Cliente envia um HELLO (1) para o servidor, uma mensagem de 2 bytes
2. Ao receber o HELLO (1), o servidor envia uma mensagem CONNECTION (2) para o cliente, composta de 2 bytes para o tipo da mensagem e 4 bytes para a porta

3. Ao receber do servidor um CONNECTION (2), o cliente envia uma mensagem INFO FILE (3), composta de 2 bytes para o tipo da mensagem, 15 bytes para o nome do arquivo, 8 bytes
4. Quando o servidor recebe uma mensagem INFO FILE (3), os dados de nome de arquivo e seu tamanho são salvos num dicionário utilizando como chave o IP:PORTA da conexão. Dessa maneira é possível acessar os dados posteriormente. Em seguida, o servidor envia ao cliente uma mensagem de OK (4).
5. Ao receber um OK (4) do servidor, o cliente envia ao servidor uma mensagem FIM (5) e parte para a transferência do arquivo.

Tanto no servidor como no cliente foram utilizados métodos do módulo nativo *struct*, para converter de inteiro para bytes (e vice-versa) na quantidade correta de bytes e utilizando a notação *big-endian*. Para *strings*, utilizou-se dos métodos *encode* e *decode*, sempre utilizando ASCII.
Com a lógica implementada partiu-se para a transferência do arquivo via UDP.

2.2 Envio do arquivo

Nesse ponto faltava realizar a transferência do arquivo. De maneira análoga ao passo anterior, no cliente e no servidor foi criado um *socke*t, mas dessa vez utilizando SOCK_DGRAM para definir que é UDP.
No lado do cliente, o arquivo é lido de 1000 bytes em 1000 bytes, uma vez que esse é o tamanho máximo permitido em cada pacote e é enviada uma mensagem FILE (6) no formato descrito na especificação: 2 bytes para o tipo mensagem + 4 bytes para o número de sequência (que se inicia com 0 e é incrementada conforme enviamos pacotes) + 2 bytes o tamanho do que é enviado + até 1000 bytes do arquivo. No envio do arquivo, como não foi implementada nenhuma política de janela deslizante, o que é realizado é apenas verificar se a mensagem recebida imediatamente após o envio a confirmação de envio esperada. Se sim, partimos para o envio do próximo pacote e caso contrário enviamos novamente o pacote atual.
No lado do servidor, uma *thread* controla o recebimento via UDP, tenta obter os dados e envia uma confirmação (ACK) caso tudo esteja correta. No fim, teremos o arquivo salvo na pasta *saida*.

2.2.1 Janela deslizante

Conforme especificado, era necessário implementar algum mecanismo de janela deslizante que tratasse melhor perdas de pacotes. Durante o desenvolvimento, tentou-se implementar o método GO-BACK-N mas algumas dificuldades foram encontradas e não foi possível realizar a implementação em tempo hábil para entrega.

3. Desafios, dificuldades e imprevistos do projeto

A implementação do trabalho como um todo foi um grande desafio, uma vez que os conceitos de UDP, transferência de arquivos e janelas deslizantes não é tão simples de ser entendida.
A principal dificuldade enfrentada foi na implementação da janela deslizante. Bastante tempo de estudo foi empregado e ainda assim conforme dito não foi possível implementar em tempo hábil.

5. Conclusão

Os programas implementam bem o protocolo de mensagens TCP que foram definidos e o upload de arquivos funciona bem, ao menos nos testes realizados. Ainda assim, ficou pendente a implementação da janela deslizante.

Referências

Python. Docs. Disponível em: <https://www.python.org/doc/>. Acesso em: 03 jul. 2019.
Aulas da disciplina “Redes de Computadores”. UFMG Virtual. Disponível em: <virtual.ufmg.br>. Acesso em: 14 jan. 2021.