TP2: Serviço de transferência rápida e fiável de dados sobre UDP

Comunicações por Computador
Universidade do Minho

André Peixoto
Filipe Cunha
João Monteiro

Conteúdo

1 Introdução

Este trabalho teve como objetivo implementar uma aplicação de transferência rápida e fiável de dados sobre UDP, o qual foi dividido em 2 fases.

Neste relatório, especificamos o nosso protocolo, assim como a implementação da nossa solução que foi desenvolvida ao longo de 6 aulas.

2 Especificação do protocolo

Para o protocolo de transferencia, a comunicação é realizada via UDP e pela porta 7777.

2.1 Formato PDU

Existem 3 tipos de PDU: de dados, de Syn e de SynAck/AckSynAck (download ou upload).

O formato (geral) escolhido para o PDU foi o seguinte: numero de sequencia (nseq): inteiro correspondente à ordem do primeiro octeto de dados no segmento; número de ack (ackNseq): inteiro que corresponde à ordem do octeto seguinte na sequência; dados (data): array que contém cada segmento dos dados a transferir; flag: inteiro definido para indicações específicas; filename: String contendo o nome do ficheiro a transferir; crcValue: inteiro longo que corresponde à deteção de erros na mensagem.

2.2 Interações

O servidor tem uma thread que recebe o PDU syn e mal o recebe, cria uma nova thread, Receiver, a qual vai ter outro socket. Essa nova thread fica a escutar o cliente e quando precisa de enviar algo para o cliente, essa thread cria ainda outra Thread, Thread Server Sender, sendo a Thread responsável para o envio da resposta ao cliente. Desta forma, deixa-se sempre o socket principal de início de conexões vazio para atender varios clientes ao mesmo tempo, onde cada cliente fica atendido por 2 threads: uma para enviar e outra para receber. O socket UDP tem o problema de que caso chegue um pacote seguido de outro, substitui o primeiro, assim com a abordagem acima referida, deixa-se os sockets mais livres possíveis (cada Thread é responsável por um socket).

3 Implementação

A implementação foi executada em Java, com o auxílio das classes DatagramSocket, para utilização dos pacotes UDP, bem como Socket e ServerSocket para pacotes TCP (por serem mais fiáveis numa conexão cliente-servidor para reduzir ou eliminar perdas de pacotes).

3.1 Classes relevantes

Segue-se uma breve discrição das classes mais relevantes do código:

- **AgenteUDP**: classe que comunica enviando e recebendo datagramas UDP e sobre o qual assenta toda a comunicação

- **TransfereCC**: componente que implementa o transporte fiável e seguro de dados

- **TServerReceiver**: thread criada pelo servidor principal quando recebe um PDU de syn, mantendo o servidor principal livre

3.2 Bibliotecas usadas

A seguir encontram-se as principais bibliotecas utilizadas:
java.nio.file.Path;
java.nio.file.Paths;
java.nio.file.Files
java.net.DatagramPacket
java.net.DatagramSocket
java.util.concurrent.locks.ReentrantLock
java.io.BufferedReader
java.io.BufferedWriter
java.util.concurrent.locks.Condition;
java.util.concurrent.locks.Lock;
java.io.InputStreamReader
java.io.OutputStreamWriter
java.io.PrintWriter
java.io.FileWriter
java.net.Socket
java.net.ServerSocket;

4 Testes e Resultados

Os testes foram realizados no terminal e da seguinte forma:
- download e upload de 2 ficheiros de tamanhos diferentes por um cliente em localhost (no mesmo pc);
- download e upload desses mesmos 2 ficheiros por um ou vários clientes conectados à mesma rede.

Abaixo, segue-se a exemplificação dos comandos, considerando o download do ficheiro file2.txt em localhost:

Servidor: Java MainServer 7777
Cliente: java Client download file2.txt localhost 7777

5 Conclusões

Com a realização deste trabalho, consolidámos os nossos conhecimentos relativos à camada de transporte. Fazendo uma apreciação final do nosso projeto, estamos confiantes que conseguimos alcançar os objetivos propostos de criar um servico rápido e fiável de transferência de dados sobre UDP, pois cremos que atingimos as funcionalidades essenciais e alguns dos extras requeridos pelos professores com uma estrutura de código bastante otimizada.