Redes de Computadores

LEIC-T

2016/17

Guia de Laboratório 3 – Programação com sockets, camada rede e protocolo IP

Objectivos

Este guia tem por objectivo a aquisição de conhecimentos sobre a programação de aplicações sobre redes usando a interface de *sockets* e a familiarização com alguns aspectos importantes do funcionamento da camada de rede e do protocolo IP.

Este guia diz respeito a 2 aulas de laboratório. O programa a desenvolver, descrito no exercício 1, será entregue, através do Fenix, até ao dia 18 de Novembro. Na semana seguinte, durante a aula e em simultâneo com o Guia 4, será realizada uma demonstração da aplicação e será solicitada a realização de uma pequena modificação ao programa.

Exercício 1 – Programação com Sockets

Nota prévia: recomenda-se que comecem por experimentar o exemplo com sockets TCP que se encontra no livro da cadeira, em Java (5ª edição) ou Python (6ª edição) conforme a linguagem que preferirem usar. A explicação é dada considerando Java mas podem optar por Python.

Este exercício consiste na programação de uma aplicação tipo Dropbox, chamada *Caixote*. A aplicação é baseada no modelo cliente-servidor. Existe um servidor centralizado que guarda uma cópia dos ficheiros do utilizador e uma aplicação cliente que é executada pelos utilizadores para sincronizar o conteúdo de uma directoria local com o servidor. Toda a comunicação é feita sobre TCP/IP.

Caixote

Este é o programa cliente executado por um utilizador para aceder ao serviço. O programa é executado correndo *java Caixote hostname port username directoria*, sendo *Caixote* o nome do programa/classe, *hostname* o nome ou endereço IP da máquina onde é executado o servidor, *port* o porto em que o servidor está à escuta, *username* a identificação do utilizador e *directoria* o nome da directoria a sincronizar.

Quando o programa arranca, estabelece uma conexão TCP com o servidor e realiza a sincronização. Enquanto é executado, programa tem de ir apresentando informação sobre o progresso da sincronização ou uma mensagem de erro em caso de insucesso. Se a conexão com o servidor falhar deve também ser apresentada uma mensagem que explique o que sucedeu.

O processo de sincronização consiste em realizar cópias de ficheiros e directorias, entre o cliente e servidor e vice-versa, de forma a que no final ambos tenham os mesmos ficheiros/directorias e a mesma versão dos ficheiros. Uma versão de um ficheiro é considerada a mais recente se tiver a data de modificação mais recente. Assumimos que clientes e servidor têm os relógios sincronizados.

Para simplificar, no *Caixote* assumimos que os ficheiros não são apagados. Se um ficheiro existe num local e não no outro, deverá ser copiado para o local onde não existe. Também assumimos que os ficheiros não são modificados externamente durante o processo de sincronização.

Servidor

É executado correndo *java Server port*, sendo *Server* o nome da classe Java que o implementa e *port* o número do porto TCP em que fica à escuta. Tem de suportar ligações simultâneas de vários clientes, logo tem de correr várias *threads.* No entanto, o servidor apenas suporta a realização de um processo de sincronização da mesma directoria de cada utilizador de cada vez. Caso o mesmo utilizador já esteja a sincronizar uma directoria, outros pedidos de sincronização da mesma directoria são rejeitados.

O servidor apenas mantém informação sobre os ficheiros, nomeadamente:

- O nome e *path* relativo do ficheiro.
- O conteúdo do ficheiro.
- A data de modificação do ficheiro.
- A quem pertence o ficheiro.

Toda esta informação pode ser guardada no sistema de ficheiros. Recomenda-se a criação de directoria com nomes do tipo *username-nomeDaPasta*. Este par é sempre único. Se preferir, pode assumir que os ficheiros têm nomes simples, nomeadamente, que não incluem o caractere '-'. Os utilizadores não têm credenciais de acesso.

Teste

Os programas têm de ser testados. A forma mais simples de o fazer consiste em executá-los num único computador indicando *localhost* ou *127.0.0.1* no parâmetro *hostname* dos clientes. No entanto, uma vez bem testado desta forma deve ser também testado correndo o servidor numa máquina e os clientes em outra(s).

O *wireshark* pode ser uma poderosa ferramenta de *debug*.

O comando *stat* dá informação relevante sobre um ficheiro.

Protocolo

Deverão começar por desenhar o protocolo de comunicação: quais as mensagens, qual o formato de cada mensagem, o que fazer ao receber cada uma das mensagens, etc.

Deverá ser entregue um documento, em PDF, com a descrição do protocolo e formato das mensagens. Este documento deve ser muito sucinto, de preferência de 1 só página. Não é pedido um relatório.

Exercício 2 – Tabela de encaminhamento em Linux

O sistema operativo Linux tem uma tabela de encaminhamento (*forwarding/routing*) que usa para determinar para que interface deve encaminhar cada datagrama IP. Neste exercício vamos observar o conteúdo dessa tabela. Pode ser útil consultar o manual do comando *ip route* executando *man ip route* ou acedendo a http://www.die.net/.[1]

1. Execute o comando *ip route* num terminal.
2. Quantas linhas tem a tabela?
3. O que significa cada uma dessas linhas?
4. Qual é a sub-rede a que está ligado o computador? Dê a resposta no formato CIDR.
5. Qual é o endereço IP da *default gateway*, ou seja, da interface do *router* ligada a essa sub-rede?

1Em Windows o comando *route print* fornece informação semelhante mas num formato completamente diferente, de modo que não é recomendado o seu uso neste laboratório. Em GNU/Linux também existe o comando *route*, mais antigo.

Exercício 3 – Comando *traceroute* e cabeçalho IP

Este exercício começa com a utilização do comando *traceroute* como ferramenta para a determinação de caminhos percorridos por pacotes em redes IP. Pode ser útil consultar o manual do comando *traceroute*, por exemplo, executando num terminal o comando *man traceroute* ou acedendo a http://www.die.net/. Responda às seguintes questões:

1. Que tipos de pacotes o *traceroute* usa para efectuar a descoberta de caminhos?[2]
2. Que tipos de pacotes são usados como resposta pelos nós ao longo do caminho (incluindo o último)?

Neste exercício é efectuada uma captura de pacotes que resultam da execução do comando *traceroute* para análise dos campos *TTL* e *Protocol* do cabeçalho IP desses pacotes (ver Referências no final deste documento).

Execute os seguintes passos:

- Iniciar uma captura de pacotes utilizando o Wireshark;
- Emitir o comando *traceroute www.ietf.org*
- Terminar a captura de pacotes;
- Aplicar um filtro adequado à informação que pretende visualizar no Wireshark.

Responda às seguintes questões:

3. Quantos *hops* foram observados?
4. Qual o último nó em Portugal que se encontra no caminho até ao *host www.ietf.org*? Pode obter a localização geográfica da interface com um endereço IP usando p.ex. o *site* http://www.geoiptool.com/.
5. Qual a razão pela qual por vezes não é mostrada informação relativa a alguns nós (são mostrados asteriscos)?
6. Seleccionar o primeiro pacote enviado pelo seu equipamento terminal para descobrir o caminho. Observar os detalhes do protocolo IP. Qual o valor do campo *Protocol* e o seu significado?
7. Qual o valor do campo *TTL* desse mesmo pacote e o respectivo significado?
8. Seguidamente observar o segundo pacote enviado para o mesmo fim. Indicar os campos do cabeçalho IP que sofreram alterações.
9. Observar agora o quarto pacote enviado para o mesmo fim. Dizer quais os campos do cabeçalho IP que desta vez foram alterados.
10. Este último pacote foi fragmentado? Justifique a sua resposta.
11. Qual o tamanho em *bytes* do cabeçalho destes pacotes?
12. Qual o tamanho em *bytes* do campo de dados desses mesmos pacotes?
13. Quantos pacotes são enviados para a descoberta de cada nó ao longo do caminho?
14. Por qual motivo são utilizados mais do que um pacote para a descoberta de cada nó ao longo do caminho?

Exercício 4 – Fragmentação de datagramas IP

O objectivo deste exercício é observar a fragmentação de pacotes IP que resulta do envio de um pacote cuja dimensão é superior ao tamanho máximo permitido pela interface de rede.

Para o efeito, execute os seguintes passos:

- Iniciar a captura de pacotes em modo não promíscuo utilizando o Wireshark;

2Em Windows o comando é designado *tracert*.

- Emitir o comando *ip link*[3] e observar o tamanho máximo para um pacote enviado pela interface de rede (MTU);
- Enviar um pacote de *ping* com 50B, executando o comando: *ping -c* 1 *-s* 50 *tagus.inesc-id.pt*
- Enviar um pacote de ping com 4449B, executando o comando: *ping* -c 1 -s 4449 *tagus.inesc-id.pt*
- Terminar a captura de pacotes;

Responda às seguintes questões:

1. O primeiro pedido de *ping* foi fragmentado? Justifique a sua resposta.
2. O segundo pedido de *ping* foi fragmentado? Justifique a sua resposta.
3. Em quantos pacotes foi dividido o pedido fragmentado?
4. Existe a possibilidade de saber, antes de os receber na totalidade, em quantos fragmentos vem repartida uma mensagem IP? Justifique a sua resposta.
5. Quais os campos do cabeçalho IP que permitem a reconstrução do pacote original a partir dos fragmentos?
6. Determinar a dimensão mínima a ser atribuída ao comando *ping* para que o pacote seja fragmentado?
7. Indicar em quantos fragmentos é dividido um pacote IP com 2981B?

Exercício 5 – Problema prático

Um router tem as seguintes quatro entradas na sua tabela de expedição (*forwarding*): (5.55.0.0/19, A); (5.55.16.0/21, B); (5.55.4.0/22, C); (5.0.0.0/8, D). Em cada par, o primeiro campo designa um bloco de endereços IP e o segundo uma interface do router. Indique justificando por qual das interfaces é expedido um datagrama com endereço de destino:

1. 5.55.12.1
2. 5.55.20.1
3. 5.6.12.1
4. 5.55.128.0

Referências

[1] Internet Protocol - IP: http://www.ietf.org/rfc/rfc791.txt

[2] Capítulo 4, do Livro “Computer Networking: A Top Down Approach”, James Kurose & Keith Ross, 6ª Edição, 2012

3Um comando alternativo será *sudo ifconfig*, já mais antigo.