Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

TP2 - Servidor DNS

Vinicius Julião Ramos
2018054630
viniciusjuliao@dcc.ufmg.br

4 de Outubro de 2020

1 Introdução

O objetivo desse trabalho é implementar um servidor DNS que tenha uma interface de linha de comando. Esse serviço é implementado com o protocolo UDP e para fornecer um servidor que esteja em execução ao mesmo tempo que o leitor de linha de comando, dividiu-se execução em dois subprocessos. Tais subprocessos foram criados através da implementação de *threads*, sendo que um deles realiza a "escuta"da rede, enquanto o outro recebe dados pela linha de comando. A parte que recebe requisições de rede realiza verifica se o endereço está presente em seus dados, e caso não esteja, requisita o endereço para os outros servidores DNS que são adicionados como forma de *links*.

2 Decisões de implementação

Na solução apresentada por esse trabalho, um mesmo programa se encarrega de executar cliente e servidor UDP. A comunicação é feita de forma que a parte servidor é requisitado para responder se conhece algum endereço IP de acordo com o *hostname*. No caso em que o programa não conheça o nome do *host*, uma parte cliente é chamada para requisitar a outros servidores DNS sobre tal *hostname*.

2.1 Protocolo de comunicação

Uma vez que protocolo de comunicação, foi descrito pela especificação do trabalho, esse relatório descreverá as decisões de implementação. Entretanto há três pontos que merecem atenção quanto ao padrão de comunicação:

1. As mensagens trocadas entre servidor e cliente possuem um cabeçalho de 1 Byte, devendo conter o número inteiro `1` ou `2`. Esse valor não se trata de um *string* contendo o caractere `'1'`, uma vez que esse valor na tabela ASCII seria convertido para o inteiro `49`. O mesmo acontece com o código `2`.

2. Quando o servidor não encontra um endereço para determinado host, inclusive analisando os *links*, deve-se retornar o *payload* contendo `-1`. Nesse caso o corpo da mensagem possui uma *string* `-1"` e não um valor inteiro. Essa decisão foi tomada com base numa postagem no fórum de dúvidas da disciplina no Moodle.

3. O endereço IP que o servidor envia para o cliente é uma *string* em formato decimal. Ou seja, dado um endereço `192.0.0.1`, o servidor envia uma mensagem com a *string* `"192.0.0.1"`.

2.2 Estruturas de dados

Uma vez que a solução foi elaborada em C++, as estruturas utilizadas foram providas pela biblioteca padrão da linguagem. Os *hostnames* e seus respectivos endereços são salvos em `std::map` [1] em que as chaves são os nomes e o valor é o endereço IP correspondente. Esse detalhe de implementação faz com que a busca pelo endereço de um *host* tenha um custo menor em contraste ao uso de um *array*.

Também era necessário armazenar os links que futuramente seriam utilizados para encontrar os *hostnames* que não estivessem presentes no mapa. Então, criou-se uma *struct* que armazena duas strings: endereço IP e porta UDP em que outro servidor DNS estiver em execução. Como podem existir múltiplos links, novamente um recurso da biblioteca padrão de C++ foi utilizado, provendo um `std::vector` [2] que permite o armazenamento em forma de um *array* dinâmicamente gerenciado.

Tais estruturas instanciadas em forma de ponteiros, para que os dois subprocessos possam acessá-las. Portanto, há um trecho de memória comum a mais de uma função, sendo que apenas a parte que recebe comandos pelo terminal podem alterar as informações dessas estruturas. Já o processo servidor utiliza o mapa e o vetor apenas como consulta que é igualmente executada pelo comando *search* da linha de comando. Na thread que executa um socket ativo, existe uma instância da classe **UDPServer**, a qual é responsável pelo recebimento das requisições de *search*. Essa classe implementa as funções de socket passivo POSIX.

2.3 Comando *search*

Esse comando merece atenção pelo uso da classe **UDPClient**. Neste ponto, a primeira etapa é validar junto ao mapa de *hosts*, se o nome pesquisado está presente ou não. Uma vez que o host não é encontrado, então a segunda etapa se trata da criação de múltiplas instâncias de **UDPClient** para requisitar a cada servidor DNS (link) se estes possuem o *hostname* pesquisado. Tanto o subprocesso servidor quanto o subprocesso de comandos da linha de comandos podem executar um *search*.

3 Execução

Visto que o código foi escrito em C++, a compilação é feita por meio de um arquivo *Makefile*. Então, usando uma máquina Linux, execute:

```
foo@bar:~/tp2$ make
```

Após a compilação, um arquivo executável será gerado (**servidor_dns**). O comando de execução é:

```
$ ./servidor_dns <porta udp> [arquivo-inicial]
```

O argumento obrigatório `<porta udp>` é o número da porta que o servidor dns alocará. Já o `[arquivo-inicial]` é um argumento opcional no qual, dado o arquivo nomeado por esse parâmetro, o programa executará os comando presentes nele. Após a execução dos comandos do arquivo, novos comandos poderão ser digitados no terminal.

Um aspecto que merece atenção é que a versão do IP é parametrizado por um atributo da classe **Utils**. Caso seja necessário a mudança da versão do protocolo pode ser alterada nas linhas iniciais do arquivo de código `main.cpp`

4 Conclusão

O trabalho prático foi útil para visualizar e entender a comunicação UDP e também entender porquê o serviço de DNS utiliza tal protocolo. A utilização de *threads* e de estruturas de dados que compõem uma memória compartilhada entre processos, mostrou a relação próxima entre aplicações de rede e múltiplos subprocessos.

[1] http://www.cplusplus.com/reference/map/map/
[2] http://www.cplusplus.com/reference/vector/vector/

Ao tentar executar esse trabalho juntamente com o de outro colega de classe, a implementação correta do protocolo se mostrou eficaz. Uma vez que os dois trabalhos estavam de acordo com a especificação, os programas conseguiram se comunicar perfeitamente.