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 Protocolo de comunicação

Cada um dos programas, cliente e servidor, possui uma forma particular de tomada de decisões, um algoritmo, no qual, de acordo com o cabeçalho da mensagem recebida, deverá tomar determinadas ações. Apesar do comportamento diferente entre os programas, estes são complementares, logo compõem um protocolo de comunicação.

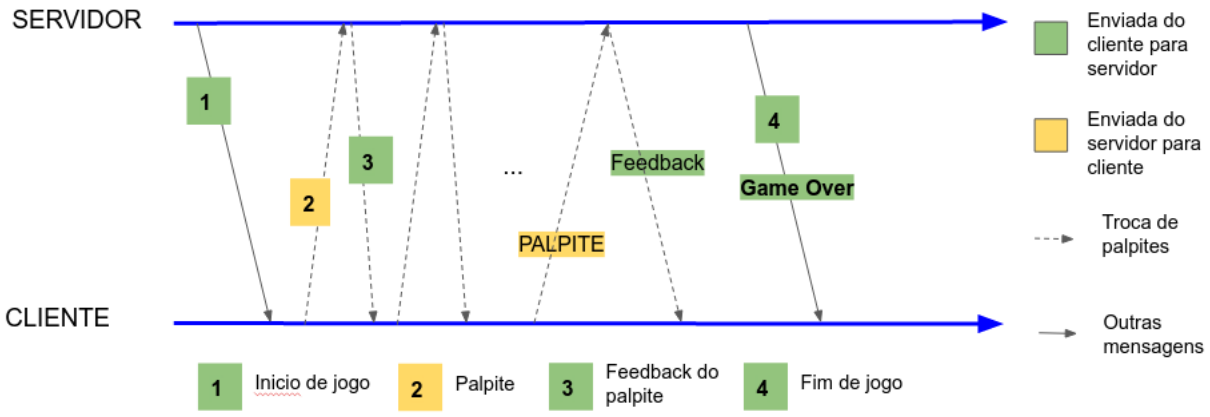

Figura 1: Protocolo de comunicação entre cliente e o servidor.

O diagrama da Figura 1 exibe como se dá a comunicação entre os programas, sendo que um ponto merece atenção. Após o cliente identificar que acertou toda a palavra, podem ocorrer diversas requisições entre **Palpites** e **Game Over**, mas a conexão só é encerrada após o envio da mensagem de fim de jogo.

Após o estabelecimento da conexão, a primeira mensagem é enviada pelo servidor, sendo que o segundo passo é composto pelo processo de recebimento de palpites. Nesse ponto, o servidor só aceitará requisições que têm o identificador de **Palpite** no cabeçalho, até que toda a palavra seja acertada. Por fim uma mensagem indicando o fim de jogo é enviada e a conexão é fechada.

Após finalizar a conexão com um cliente, o servidor poderá receber outro cliente que deseje jogar o jogo. Então a cada nova conexão o jogo é reiniciado.

3 Implementação

Dadas as especificações do trabalho, necessitou-se organizar o código e implementar funcionalidades que viabilizassem a boa execução dos programas. Dentre tais decisões destaca-se o uso de um *array* e de um *map* para armazenar a palavra do jogo e computar quais os caracteres foram recebidos como palpites no lado servidor. Optou-se pela utilização de um *map* para que o custo de busca de um caractere na palavra seja da ordem de $O(logn)$ com n sendo o tamanho da palavra. É sabido que tal implementação não reduz o custo geral de execução, uma vez que a busca pelos índices os quais o caractere está inserido na palavra é da ordem de $O(n)$, dada a necessidade de varrer toda a *string*.

Na parte cliente, utilizou-se de duas estruturas aninhadas, uma do tipo *vector* e outra do tipo *pair*, ou seja, um arranjo de tuplas. A tupla é composta por um caractere e um *booleano* em que o cliente marca nos índices corretos se o caractere está na palavra oculta. Tal marcação é feita de acordo com o *feedback* recebidos do servidor após o palpite. Por fim, ao final de cada iteração para realizar o palpite, o cliente valida se a palavra foi completamente adivinhada, sendo que tal operação tem um custo de execução $O(n)$.

Para garantir a persistência da execução dos programas, alguns aspectos do protocolo foram inferidos a partir da leitura da descrição do trabalho. Tais decisões de implementação são:

1. No cliente, a mensagem que identifica fim de jogo tem alta prioridade. Logo, sempre que o cliente recebe um cabeçalho de tipo com o identificador 4, o programa cliente é encerrado.

2. Em ambos os programas, dado o *workflow* descrito na Figura 1, o destinatário da mensagem está em um laço aguardando até que a mensagem com o cabeçalho correto seja enviado. Por exemplo, no cliente, a primeira mensagem recebida deve ter o código 1 ou 4 (considerando o item acima), e até que um desses códigos não sejam recebidos, o programa permanece aguardando o cabeçalho correto.

3. Para garantir que o servidor permaneça em execução mesmo com a "queda"do cliente, um pequeno tratamento de erros foi adicionado. O servidor valida se o cliente está enviando informações ou não. Caso o cliente não esteja mais em execução, o servidor reinicia o jogo e aguarda uma nova conexão.

Por fim, ainda sobre o padrão de implementação que os programas devem ter, de acordo com a especificação deste trabalho, há mais outros dois fatores que merecem atenção: A palavra oculta do jogo é uma constante que está declarada nas primeiras linhas do código **servidor.cpp**. O mesmo acontece para a versão do protocolo *IP* utilizada; também é uma constante definida nas primeiras linhas de código do arquivo citado anteriormente.

4 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:~/tp1$ make
```

Após a compilação, dois arquivos binários serão gerados (**servidor** e **cliente**). Os comandos de execução são:

```
$ ./servidor-mt <porta tcp>
```

```
$ ./cliente-aluno <ip servidor> <porta tcp servidor>
```

Vale lembrar que, para trocar a versão do protocolo IP utilizada ou a palavra secreta do jogo, basta mudar as constantes no código de **servidor.cpp**. Então recompile usando o comando *make*.

5 Conclusão

O trabalho prático foi útil para visualizar e entender a comunicação TCP entre cliente e servidor. A utilização de estruturas de dados num cenário simples de um jogo, mostrou como uma aplicação de rede funciona e abriu horizontes para o entendimento da importância da definição de um protocolo de comunicação.

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.