Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

TP0

Monica Emediato
monica_emediato@hotmail.com
Vinicius Julião Ramos
viniciusjuliao@dcc.ufmg.br

22 de Março de 2020

1 Introdução

O objetivo desse trabalho é realizar uma simples comunicação entre programas através de *sockets* do protocolo TCP. Para isso, desenvolveu-se uma aplicação que computa a chamada de alunos em uma classe, em que há um professor e diversos alunos, através de um programa servidor e dois programas clientes (um para os alunos e outro para o professor). Essa aplicação funciona de forma que o programa servidor gera uma senha para os alunos e outra para o professor, ao executar o programa informando a senha, cada aluno envia a senha e o número de matrícula que é salvo pelo servidor. Já o professor, ao enviar apenas sua senha, recebe uma lista contendo todos os alunos que tiveram a presença registrada até o momento.

2 Diferença Semântica

Cada um dos programas cliente possui uma forma particular de comunicação, um protocolo, na qual, de acordo com a senha recebida, o servidor destina o fluxo de execução para uma comunicação entre o cliente do professor ou para o cliente do aluno; no caso de não receber alguma senha válida, a conexão é encerrada. Algumas mensagens trocadas entre os programas são padrões do protocolo, e para estes casos, ao receber uma mensagem padrão quaisquer dos programas verificam se o conteúdo do pacote recebido é realmente o que se espera. Entretanto há algumas mensagens que têm conteúdo variável, para estes casos é necessário testar se a quantidade de bytes recebida é o que se espera, ou então aguarda-se alguma mensagem para o fim da comunicação.

O diagrama da Figura 1 exibe como se dá a forma de comunicação entre os programas em que há dois detalhes que merecem certa atenção: na troca de mensagens do fluxo do professor, no caso deste trabalho, implementou-se de forma que o servidor envia uma *string* contendo um número de matricula seguido por um caractere **\n** em cada envio, ao final do envio da lista há o envio do caractere **\0**, identificando o fim da lista. O fechamento da conexão é realizado após o envio de uma mensagem "OK"pelo cliente. Vale lembrar que o maior inteiro de trinta e dois bits (quatro bytes) é o número 4294967295, que possui dez caracteres em sua representação *string*, logo, são enviados de onze em onze bytes, no máximo, sendo que tal mensagem tem tamanho variável, dependendo da quantidade de bytes necessária para representar tal inteiro de forma decimal.

No segundo caso, na troca de mensagens entre o cliente aluno a informação contendo o número de matrícula do aluno possui conteúdo variável, mas o tamanho é fixo de quatro bytes, o que representa um inteiro de trinta e dois bits. O tamanho de tal mensagem é validado sempre que recebida, e computa-se o número de matrícula apenas após o recebimento dos quatro bytes da matrícula, seguidos do envio com sucesso da mensagem "OK"pelo servidor; sendo que essa ultima mensagem identifica o final da comunicação.

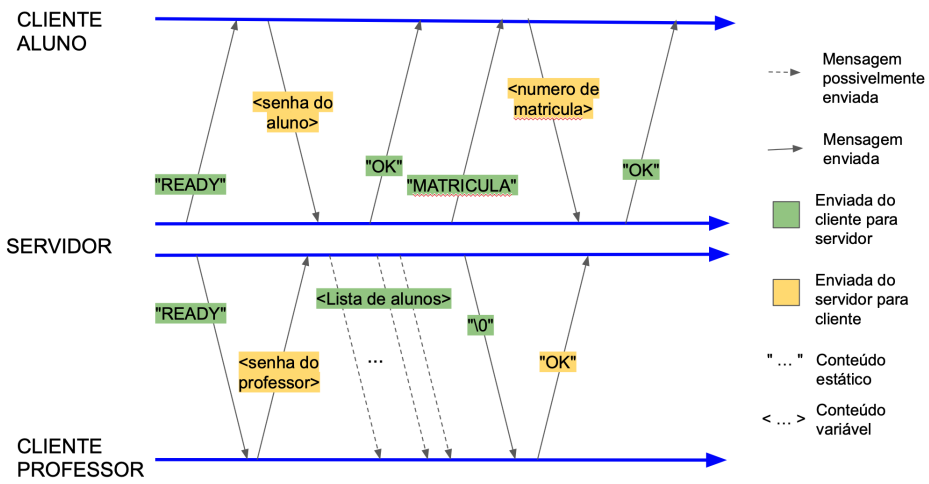

Figura 1: Protocolo de comunicação entre clientes e o servidor.

3 Tratamento de múltiplas conexões

As múltiplas conexões foram tratadas de duas maneiras diferentes, e para isso desenvolveu-se dois tipos de arquivo servidor diferentes; um que utiliza de *threads* e outro não. No caso do servidor que não utiliza *threads* (arquivo **servidor.c**), as múltiplas conexões são tratadas de forma que se permite dez conexões simultâneas, entretanto, o servidor responde apenas uma por vez de modo que após receber uma primeira comunicação com determinado cliente, a segunda conexão só iniciará a troca de mensagens após a finalização da primeira, criando uma fila de prioridades na qual a função *listen()* recebe como parâmetro a quantidade de elementos (conexões) que essa lista terá.

Quanto ao caso do servidor multi processos (arquivo **servidor-mt**) a função *listen()* também implementa dez conexões para a fila de prioridades, porém ao utilizar a execução paralela de conexões, em um *poll* de tamanho cinco, pode-se comunicar-se simultaneamente com cinco clientes enquanto outros cinco aguardam na fila de prioridades, até que haja uma "vaga"no *poll* e assim serão executados.

4 Implementação e Execução

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* com duzentos e cinquenta e seis posições para armazenar os alunos, ou seja, o número máximo de alunos que podem ser registrados é 256. Esse número foi acertado com o professor em sala de aula para que os alunos se prevenissem de implementar a estrutura de lista encadeada. Quanto à geração automática das senhas de acesso ao servidor, as senhas são compostas de algarismos alpha numéricos, valendo destacar que as senhas são *case sensitive* das quais a primeira cadeia de 8 bytes impressa é a senha do professor e a última é a senha dos alunos.

Já na implementação do **cliente-aluno**, acrescentou-se um novo parâmetro de execução que representa o número de matrícula do aluno, dada a não especificação na descrição do trabalho, optou-se por permitir que cada aluno realize a chamada passando seu próprio número de matrícula. Caso tal parâmetro não seja passado, ou algum outro parâmetro, então o programa retornará uma mensagem de erro. Essa condição dos parâmetros é válida para todos os programas desenvolvidos.

Da **execução** dos programas, antes de qualquer chamada dos programas, deve-se executar o comando **make**, e então quatro arquivos executáveis serão gerados (**servidor**, **servidor-mt**, **cliente-professor**, **cliente-aluno**), sendo um destes o arquivo *multi threads* possuindo o sufixo "mt". a chamada desses aquivos se dará da forma como descria na documentação do trabalho, mas vale destacar que a chamada de qualquer um dos arquivos servidor resultará na impressão de duas senhas, a primeira representando a senha do

professor e a segunda a senha dos alunos.

Sobre a execução do programa **servidor-mt**, essa se dará da seguinte forma:

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

na sequência do comando acima haverá a impressão das senhas, como já especificado anteriormente.

Quanto à execução do programa **cliente-aluno**, basta adicionar um ultimo parâmetro comparando-se com a forma de execução solicitada pela descrição do trabalho:

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

Quanto às demais execuções, segue-se da maneira que foi solicitada pela descrição do trabalho.

5 Conclusão

O trabalho prático foi útil para visualizar a comunicação TCP entre clientes e servidor utilizando dois tipos diferentes de tratamento de múltiplas conexões, além de desenvolver uma aplicação para ser utilizada em rede, que exigiu uma implementação muito bem organizada das funções e procedimentos.

Com uma pequena execução de duzentos e cinquenta chamadas em paralelo da aplicação cliente, primeiro para o servidor multi tarefas e na sequencia para o servidor com um tratamento mais básico de conexões, pôde-se observar que a quantidade de comunicações sem sucesso foi muito inferior para o caso do servidor que utiliza *threads*. Tal fato se explica pela liberação de novas conexões mais rapidamente do que para o caso em que a execução é sequencial e são tratadas apenas um cliente por vez.