

Trabalho Prático 0

Douglas W Lopes e Lucas Pereira Monteiro

05 de Outubro 2015

1 Introdução

Com intuito de evitar que alunos enviem trabalhos incorretos, uma aplicação em rede é criada nesse trabalho, onde aqueles alunos que acertarem um desafio matemático enviado pelo servidor, serão considerados aptos para fazer o upload de tais trabalhos.

Problemas matemáticos simples, como: adição, subtração, multiplicação e exponenciação serão propostos assim que o aluno, representado pela máquina cliente, conectar ao servidor (responsável por enviar tais desafios). Recebendo o desafio, o aluno deve respondê-lo, caso esteja correto, o acesso do upload estará autorizado. Caso a resposta esteja errada, novos desafios serão gerados pelo servidor e enviados ao aluno, até que ele acerte ou aperte control-c.

Aspectos técnicos importantes se devem ao fato de o cliente conectar a um endereço, sendo ele o nome ou IP e porta disponibilizada pelo servidor, no início da conexão passiva. Esses detalhes serão dados no decorrer da documentação. Outra proposta é que temos tamanhos máximos de bytes de envio de mensagens do servidor para o cliente e vice e versa, o que deixa o trabalho ainda mais desafiador.

Abaixo (Figura 1) está representado a Máquina de Estados das trocas de mensagens desse projeto. As mensagens enviadas do servidor para o cliente terão 6 bytes, as respostas dos clientes terão 3 bytes, sendo que essa resposta pode estar certa ou errada, onde a resposta estando errada novos desafios serão propostos e o ciclo começa novamente. As mensagens que informarão se a resposta do cliente está certa ou errada sairão do servidor com 1 byte cada uma.

Por fim, o objetivo desse trabalho é iniciar o contato com programação voltado a redes para nos dar a base dos próximos trabalhos práticos que estão por vir.

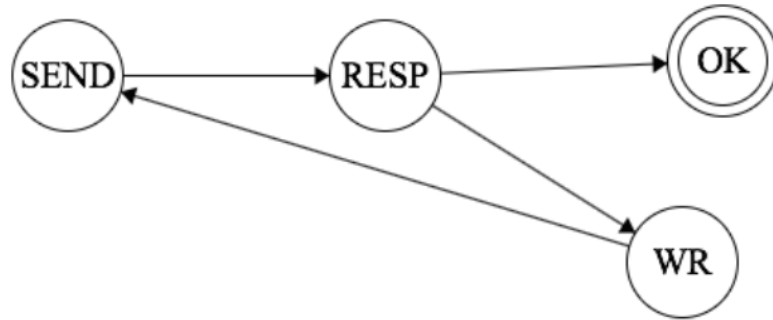


Figure 1: Representação da Máquina de Estado das trocas de mensagens desse trabalho- **Fonte:** Especificação do trabalho prático 0

2 Solução do Problema

Abaixo apresentamos as principais abstrações, em alto nível, do principal código de cada programa: servidor e cliente. Essa abstração tem por objetivo representar a solução e que essa seja de fácil entendimento. O objetivo é entender o funcionamento geral da parte mais técnica do projeto, ou seja, o algoritmo em uma linguagem de programação e aplicado na troca de mensagem cliente e servidor.

Nesse ponto há vários requisitos, e estes estão sendo vistos em salas de aula, como a ideia de sockets, por exemplo. Porém, logo numa próxima sessão alguns pontos muito importantes serão descritos aqui, para demonstrar nosso entendimento em relação ao assunto, como sockets, por exemplo.

Algorithm 1 Servidor

```
1: função SERVIDOR(Porta) Nesse ponto declaramos o nosso socket que será a
   interface da rede servidor com o cliente. Inicializamos a porta disponível e
   fazemos o "bind" do socket com o endereço e porta disponíveis. Finalmente
   permitimos por meio do "listen" que a nossa porta, através do nosso endereço
   IP, mostre-se disponível para receber uma conexão de uma máquina cliente.
   Cria-se três "buffers", um no qual o servidor envia o desafio e o cliente
   lê esse mesmo buffer (para saber de qual desafio se trata), o outro que o
   cliente envia a resposta e o servidor lê essa resposta (para comparar com
   a resposta certa) e o último buffer de "feedback" ao cliente, gerado pelo
   servidor. Após isso, seguimos o algoritmo abaixo para aceitar conexões com
   cliente e trocar mensagens:
2:   enquanto Verdade faça Conecte e troque informações com o cliente por
   meio do "Accept"
3:     enquanto Cliente não responder desafio corretamente faça
4:       Gere desafio
5:       Escreva desafio no cliente
6:       Ler resposta do cliente
7:       se Resposta do cliente está correta então
8:         Envie a confirmação da autenticação
9:       fim se
10:      se Resposta do cliente está errada então
11:        Envie ao cliente autenticação negada
12:      fim se
13:    fim enquanto
14:    Feche Socket conectado ao cliente atual
15:  fim enquanto
16: fim função
```

Algorithm 2 Cliente

```
1: função SERVIDOR(IP(ou nome),Porta) Conectar o cliente a porta correta
   do servidor,já iniciado anteriormente. Para tanto, iniciamos com um socket
   que será a interface do nosso cliente com a rede e nos conectaremos a esse e
   apartir daí temos o seguinte algoritmo:
2:   enquanto A respota desse cliente não estiver correta faça
3:     Leia Desafio vindo do servidor
4:     Apresente equação para o cliente
5:     Receba a resposta do cliente
6:     Escreva essa respota para o servidor
7:     Leia o Feedback do Servidor
8:     se Resposta do cliente está correta então
9:       A autenticação foi confirmada
10:    fim se
11:    se Resposta do cliente está errada então
12:      Autenticação negada
13:    fim se
14:  fim enquanto
15:  Feche Socket.
16: fim função
```

3 Estrutura de Dados

3.1 Servidor

- Nessa etapa serão demonstradas a utilização das estruturas do lado do servidor. Em seguida será explicitados o uso dessas mesmas estruturas do lado do cliente.

Vetores de Char - A Estrutura vetores de caracteres foi utilizada para representar os nossos buffers, uma vez que facilita a abstração de que cada mensagem de desafio deve ter 6 bytes, a resposta oriunda do cliente tem 3 bytes e o feedback 1 byte. Pois sabemos que cada campo de um veto Char tem 1 byte. Foram declarado 3 vetores: mensagem-de-resposta, mensagem-de-desafio e mensagem-feedback representador pelos buffers das mensagens relacionadas trocadas. A ideia é que quando um desafio for enviado do tamanho de 6 bytes, conforme a declaração acima; o cliente lerá um buffer de 6 bytes. Quando o cliente responder com o mensagem-resposta, logo o servidor lerá 3 bytes do buffer atual e por fim o mesmo seria feito quando o servidor enviasse o feedback ao cliente.

Equacao - Essa estrutura de dados armazena três campos: Operando 1, operando 2 e operador. Esse tipo equação armazera os operadores e operando sorteados randomicamente por funções simples de escolha aleatórias. Essa Tad tem como serviço a função: geraEquacaoAleatoria, que faz o rand dos operadores e do operando e retorna uma equação já pronta. Mas ainda precisamos adaptar o tamanho desses conteúdos desse desafio gerado e calcular a resposta antes de enviar o mesmo, para termos base de comparação para respostas corretas ou

inconrretas, explicitado em seguida.

Operando - Essa estrutura de dados é short Int (em C) e armazena os operandos das operações que serão representadas na estrutura equacao. Porém além de armazenar esse valor, operando armazena a resposta de um dado desafio com apoio da função: calcula. Esse procedimento recebe um tipo equacao e calcula qual é a resposta certa de tal desafio, além de tratar casos em que o resultado é menor que zero e mantém como resultado "correto" de 0, uma vez que não trabalharemos com resultados menores que zero. Sabendo disso, agora já podemos gerar as mensagens para enviar os desafios no seu respectivo tamanho para o cliente.

Desafio - Nessa estrutura armazenamos nossa resposta de 1 byte e nossa mensagem resposta de 6 bytes. A mensagem desafio é gerada a partir de uma função que armazena byte a byte no buffer, no caso de C são usados short int e caracteres para que essa abstração funcione e o tamanho da mensagem fique correto; em seguida inicializamos o desafio passando esses como parâmetro (na função inicializaDesafioAtravesDeMsg) colocando o código corresponde ao operando, operadores e a resposta desse desafio na estrutura Desafio. Então temos agora resposta desse Desafio e a mensagem a ser enviada para o cliente em uma estrutura de dados. A mensagem dessa estrutura de dados é escrita e enviada para o cliente pela função write das bibliotecas específicas de redes estudadas em C.

Char - Utilizamos uma variável de 1 byte chamada de feedback que, após a verificação no servidor da resposta correta pelo cliente ou não, envia o código de acerto ou erro para o cliente. Estando a resposta errada, o servidor continuará enviando desafios ao cliente.

3.2 Cliente

- Nesse ponto explicitamos como as mesmas estruturas acima são utilizadas no programa do cliente, demonstrando a capacidade de reuso dessas estruturas, sem ter de criar novas estruturas específicas para esse programa cliente.

Equacao - Essa estrutura é a primeira ser usada no programa cliente, onde essa estrutura tem um procedimento chamado: carregaEquacaoDeMsg, que recebe o buffer por onde foi enviado o desafio e o "descriptor" em um formato de equação: 2 operandos e um operador.

Operando - Uma variável do tipo operando armazenará a resposta dessa equação chegada no programa cliente, através de uma função chamado calcula que receberá a equação do tipo equacao e retornará o resultado da conta.

Desafio - Temos agora um tipo desafio com todos seus campos preenchidos, pois a função inicializaDesafioAtravesDeMsg: recebe como parâmetro a resposta do pergunta que já foi calculada, e está em um tipo operando, e o buffer no qual o desafio foi enviado do servidor. Em seguida esses campos são imprimidos na tela na ordem correta oriundos da equação que já foi formada pelo tipo: equacao.

Resposta - Temos esse tipo que contém um operando em seu conteúdo, ou seja, 1 inteiro pequeno, armazenará a resposta dado pelo cliente e o encaminhará para o servidor através de um procedimento: geraMsgDeResposta. Enviando 3

bytes para o servidor, onde o servidor lerá como resposta do cliente exatamente esses 3 bytes.

4 Algumas decisões de implementação

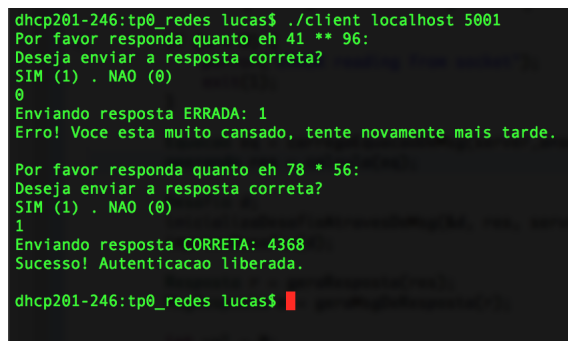
- Considerando que estamos utilizando a linguagem C, e tínhamos o desafio de manter o servidor sempre ativo e que apenas cliente fechasse a sua conexão, ou com `ctrl + c` ou acertando os desafios, tomamos as seguintes decisões:

* Criamos uma variável `bool` no servidor que cada vez chegada uma resposta oriunda do cliente, a mesma verifica se a resposta enviada é correta ou não. Se a resposta for correta enviamos o código de acerto e fechamos a conexão com tal cliente. Porém mantemos o socket inicial do servidor, anterior a de uma conexão com cliente ativo, esse nunca será fechado.

* Caso o cliente erre a questão, o socket ligando o servidor a esse cliente não será fechado e a variável `bool` continuará falsa, fazendo com um laço `"while"` continue ativo gerando desafios para esse cliente até que ele acerte ou aperte `ctrl + c`.

5 Testes

Os testes possuem algumas adições de prints para comprovar o funcionamento correto e entendimento por nossa parte, bem como algumas rotinas para decisão se o cliente enviará a resposta errada ou certa (mas que não está presente no TP entregue- isso foi feito para fins didáticos). Isso foi feito para comprovar que o nosso trabalho está em funcionamento pleno e conforme especificado.



```
dhcp201-246:tp0_redes lucas$ ./client localhost 5001
Por favor responda quanto eh 41 ** 96:
Deseja enviar a resposta correta?
SIM (1) . NAO (0)
0
Enviando resposta ERRADA: 1
Erro! Voce esta muito cansado, tente novamente mais tarde.

Por favor responda quanto eh 78 * 56:
Deseja enviar a resposta correta?
SIM (1) . NAO (0)
1
Enviando resposta CORRETA: 4368
Sucesso! Autenticacao liberada.

dhcp201-246:tp0_redes lucas$
```

Figure 2: Essa imagem demonstra a conexão de sucesso com o servidor e os desafios enviados pelo servidor e repondidos pelo cliente. Em caso de respostas erradas, o cliente só foi liberado depois de acertar algum desafio. **Fonte:** Aplicação TP0 em funcionamento.

Frizamos novamente que esses testes tem fins didáticos e que nossa intenção é facilitar o entendimento do monitor em relação ao nosso projeto. Quaisquer

```
Lucass-MacBook-Pro-3:tp0_redes lucas$ ./server

SOCKET ID: 4
Por favor responda quanto eh 41 ** 96:
tamanho de msg de desafio: 6
tamanho de msg de resposta: 3
tamanho de msg de feedback: 1

SOCKET ID: 4
Por favor responda quanto eh 78 * 56:
tamanho de msg de desafio: 6
tamanho de msg de resposta: 3
Sucesso! Autenticacao liberada.
tamanho de msg de feedback: 1
█
```

Figure 3: Essa imagem demonstra a abertura da conexão passiva e os periódicos envio de desafios pelo servidor ao cliente, até que o cliente acerte o desafio. Note que temos aqui os valores corretos de transferência de informação: 6 bytes para envio de desafio, 3 para resposta e 1 para feedback **Fonte:** Aplicação TP0 em funcionamento.

```
bash bash
dhcp201-246:tp0_redes lucas$ ./client localhost 5001
Por favor responda quanto eh 75 + 31:
Deseja enviar a resposta correta?
SIM (1) . NAO (0)
1
Enviando resposta CORRETA: 106
Sucesso! Autenticacao liberada.
```

Figure 4: Mostramos aqui a situação que o cliente acerta um desafio na primeira vez desafiado. Note que a autenticação foi liberada em seguida. **Fonte:** Aplicação TP0 em funcionamento.

```
tp0_redes - server - 80x24
Lucass-MacBook-Pro-3:tp0_redes lucas$ ./server

SOCKET ID: 4
Por favor responda quanto eh 75 + 31:
Sucesso! Autenticacao liberada.
█
```

Figure 5: Essa imagem demonstra a abertura da conexão passiva e os periódicos envio de desafios pelo servidor ao cliente, até que o cliente acerte o desafio. Nesse caso o cliente acertou na primeira vez que foi desafiado. Novamente temos o tamanho em bytes das transferências da informação. **Fonte:** Aplicação TP0 em funcionamento.

eventuais dúvidas serão esclarecidas no momento da entrevista.

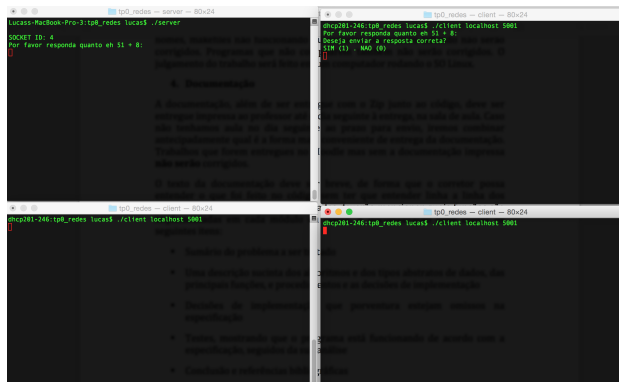


Figure 6: Nessa imagem temos a tentativa de conexão simultânea de vários clientes. Conforme requisitos não aceitaremos conexões simultâneas. Somente o primeiro conectado tem acesso aos desafios. **Fonte:** Aplicação TP0 em funcionamento.

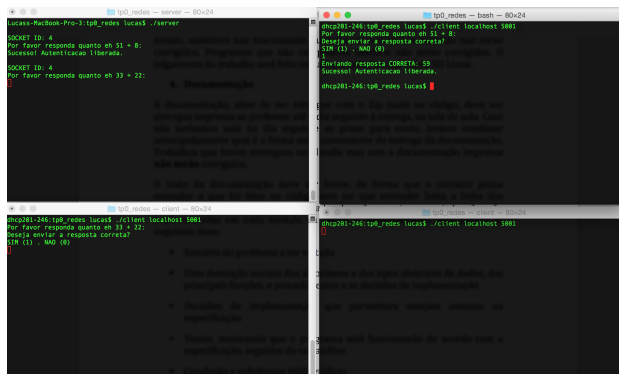


Figure 7: Por fim nesse teste, seguindo o teste da figura anterior (Figura 5), após a liberação de um cliente, o cliente subsequente é conectado e tem acesso aos desafios. **Fonte:** Aplicação TP0 em funcionamento.

6 Conclusões

Nesse trabalho, foi possível aplicar todo o conhecimento adquirido nas disciplinas de Algoritmos e Estruturas de Dados I,II,III. De vetores a estruturas mais complexas, de algoritmos simples a algoritmos complexos.

Mas é a clara a intenção desse trabalho de nos introduzir conhecimentos e aplicações básicas e iniciais de redes, como a teoria e aplicação de sockets; uso de buffer e funções para controle do tamanho da informação transmitida e toda a teoria de redes na sua comunicação, como por exemplo: o uso de portas para permissão de conexões entre redes diferentes.

Comunicação "peer-to-peer", protocolo TCP, novas bibliotecas e novas es-

truturas também foram repetidamente utilizadas. Esse trabalho nos permitiu a fixação desses, para a utilização em próximos trabalhos práticos que estão por vir, com mais facilidade.

Algumas dificuldades foram encontradas para desenvolvimento desse trabalho prático, como por exemplo: o entendimento de como a declaração de um socket, em conjunto com funções, como: "bind", "listen" e "accept" permitem a disponibilização e conexão do servidor para com o cliente. Para adquirir o verdadeiro entendimento dessa relação, foram alguns dias de pesquisa e estudos, para enfim iniciar as primeiras linhas do código. Outro ponto de dificuldade está relacionado a troca de mensagens de tamanho diferentes, pois era preciso pensar como iríamos programar de forma que a troca de mensagens acontecesse conforme os requisitos de tamanho, esse realmente foi um grande desafio e o resolvemos usando buffers para leitura e escrita do cliente e do servidor com tamanhos diferentes, conforme requisitos.

Por fim, facilidades como não permissão de conexão simultânea, o formato e estrutura dos desafios no buffer, por exemplo, facilitaram e muito o nosso trabalho. Acreditamos que o tempo fornecido para entrega do TP também foi adequado ao seu grau de dificuldade. Acreditamos que atingimos o objetivo do trabalho proposto e temos uma boa base para próximos trabalhos que estão por vir, provavelmente mais complexos e com mais detalhes de requisitos funcionais e não funcionais, entre outros. Todo o apredizado conquistado, erros e acertos serão desenvolvidos ao decorrer da disciplina por nós, e com certeza serão aplicados e trabalhos práticos cada vez melhores que serão entregues.

7 Referências

N.Ziviane, *Projeto de Algoritmos com implementação em C e Pascal*, Cengage Learning, 3ª Ed Revista e Ampliada, 2011.

T.H.Cormen, E.Leiserson, R.L.Rivest, C.Stein, *Algoritmos: Teoria e Prática*, Elsevier, Tradução da 3ª Edição ao Americana, 2012.

Sockets Tutorial: <http://www.linuxhowtos.org/C-C++/socket.html> - acessado pela última vez para esse trabalho em 05/10/2015

Materiais disponibilizados no moodle: A especificação do TP, exemplos de sockets e bibliotecas de socket.