Comunicação via Rede através de Sockets

Gabriel de Souza Ferreira – 17250447
Jean Marcelo Mira Junior – 16102369

Resumo. *O presente trabalho da disciplina de sistemas operacionais tem por finalidade trabalhar a questão de comunicação e troca de informações via rede entre computadores utilizando o conceito de sockets em C/C++.*

Palavras chave: comunicação, rede, sockets, cpp.

Introdução

Esse trabalho tem como objetivo o desenvolvimento de uma comunicação via rede entre cliente e servidor através de sockets. A funcionalidade do servidor é baseada na operação de uma loja que deseja vender apenas três itens: cerveja, água e refrigerante. Sendo assim o servidor fica esperando por conexões de clientes que desejam requisitar algum item da loja.

Discussão

A seguir, será demonstrado a lógica por trás de cada arquivo desenvolvido durante a implementação da comunicação via rede através de sockets. Vale a pena ressaltar que todo código foi devidamente comentado para o melhor entendimento e portanto este relatório exemplifica o funcionamento dos arquivos. Além de que tanto o servidor quanto o cliente tem arquivos main.cpp e portanto devem ser executados separadamente.

Main do cliente - explorando mainCliente.cpp

A mainCliente.cpp tem como principal função executar individualmente as funcionalidades do cliente, sendo ela responsável por criar, chamar a inicialização e chamar o processo de conexão com o servidor.

```cpp
int main(int argc, char const *argv[])
{
    // Construtor padrão
    Cliente c;
    // Inicializar os parâmetros do cliente na rede
    c.inicializacao();
    // Faz a troca de mensagens
    c.conexao();
    return 0;
}
```

Figura 1: Demonstração da mainCliente.cpp.

Main do servidor - explorando mainServidor.cpp

A mainServidor.cpp tem como principal função executar individualmente as funcionalidades do servidor, sendo ela responsável por criar, chamar a inicialização e chamar a função que espera pelo processo de conexão do cliente. É neste arquivo que o dono da loja ou pessoa responsável por executar o servidor deve especificar a quantidade de bebida que tem em estoque na criação do servidor. Como é possível ver na figura 2.

Dentro do arquivo há uma implementação que faz o servidor executar em loop, está implementação recebe um valor inteiro que corresponde a mensagem enviada pelo cliente, se esta mensagem for diferente da palavra “sair ” a função de conexão retorna 0 e se for igual retorna o valor -1 fazendo com que o código pare de executar.

```cpp
int main(int argc, char const *argv[])
{
    // Servidor(cerveja, agua, refrigerante)
    Servidor s(3, 3, 3); // Construtor padrão
    // Inicializar os parâmetros do servidor na rede
    s.inicializacao();
    int valor = 0; // Variável auxiliar para sair da execução do servidor

    // Executa enquanto o usuário desejar
    while (1)
    {
        // Faz a troca de mensagens
        valor = s.conexao();
        if (valor == -1)
        {
            break;
        }
    }
    return 0;
}
```

Figura 2: Demonstração da mainServidor.cpp.

Classe Rede - explorando rede.cpp e rede.h

A classe Rede é responsável por configurar os parâmetros para estabelecer a comunicação entre o servidor e o cliente.

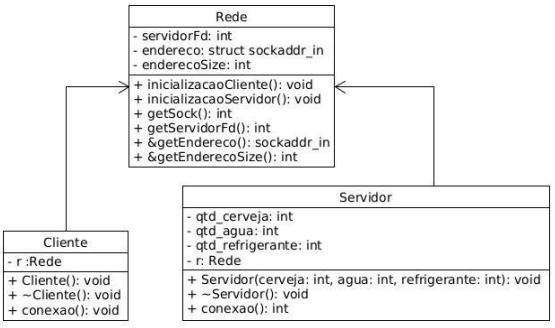

Figura 3: Relacionamento de classes..

Classe Cliente - explorando cliente.cpp e cliente.h

A classe Cliente tem como principal função enviar a mensagem do pedido via rede através de sockets. Este arquivo corresponde efetivamente às configurações do cliente, contendo funcionalidades de criação de socket, conexão de socket, conexão ao endereço IP, além de capturar e enviar ao servidor a mensagem que o usuário deseja passar a usar. A classe pode ser vista melhor na figura 4 que dispõe o diagrama UML.

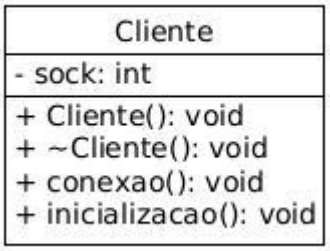

Figura 4: Diagrama UML da classe Cliente.

Classe Servidor - explorando servidor.cpp e servidor.h

A classe Servidor tem como principal função receber as mensagens/pedidos dos clientes, assim ficando em constante espera e tendo capacidade para entender o que foi solicitado. Se a bebida solicitada estiver em estoque faz a entrega da bebida, se a bebida estiver em falta informa que a quantidade solicitada está indisponível e se por algum acaso o cliente digitar algo que não corresponde com a entrada padrão para fazer o pedido informa que o nome da bebida está errado. A classe pode ser vista melhor na figura 5 que dispõe o diagrama UML.

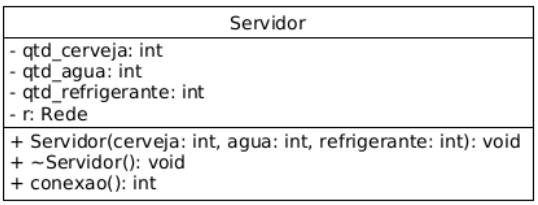

Figura 5: Diagrama UML da classe Servidor.

Conclusão

Após fazer alguns testes com a rede de comunicação implementada, foi possível ver que ela permite que vários clientes solicitem bebidas ao servidor, entretanto, notou-se que quando mais de um cliente faz o processo de comunicação com o servidor, inicia-se uma fila onde o cliente que chegou primeiro, por assim dizer, tem o seu pedido computado primeiro, mesmo que outro cliente que tenha chegado depois queira pedir.

Portanto, a comunicação proveniente da conexão é permitida, mas o servidor responde o cliente que chegou primeiro e os demais clientes têm suas respostas com uma dependência ao cliente que está na sua frente. Se por algum motivo o dono da loja quiser implementar um servidor que possibilite a retirada desta fila com prioridade de chegada, pode-se implementar o conceito de multi threads ou pelo linux usando o comando select() que manipula interrupções no envio de mensagens de comunicação entre o cliente e servidor. Nas figuras 6 e 7 pode se ver exemplo de execução do cliente e do servidor.

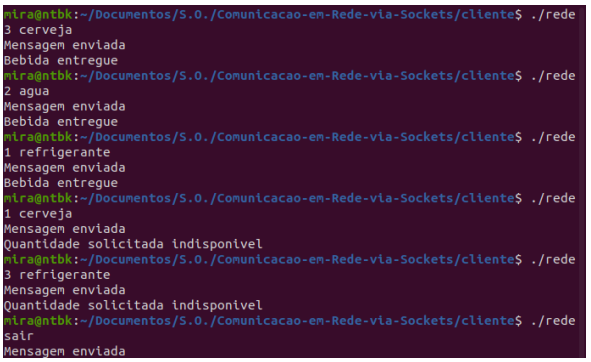

Figura 6: Exemplo execução cliente.

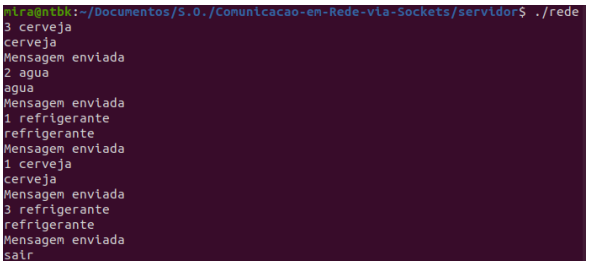

Figura 7: Exemplo execução cliente.