Nome : Riquelme Batista Gomes da Silva

Matrícula : 2021014317

Introdução:

A arquitetura cliente e servidor é amplamente utilizada em sistemas web para permitir que os usuários naveguem por diversas aplicações com diferentes propósitos. Nesse sentido, é fundamental que um profissional de tecnologia compreenda como a comunicação entre cliente e servidor é estabelecida e como os protocolos de comunicação funcionam e são criados. Nessa perspectiva, foi proposto um trabalho cujo principal objetivo é criar uma arquitetura cliente e servidor, bem como um protocolo de comunicação, para registrar, consultar, alterar e remover informações de sensores de diferentes tipos. Essas informações incluem ID, corrente elétrica, tensão elétrica e eficiência energética.

Com base nas informações fornecidas pelo cliente, o servidor pode armazená-las e realizar cálculos com elas, como, por exemplo, o cálculo da potência elétrica, que é obtida pelo produto da corrente elétrica e da tensão elétrica. Dessa maneira, o presente relatório tem como objetivo explicar como foi desenvolvido esse sistema denominado 'Smart Grid'."

Arquitetura:

A arquitetura desenvolvida é composta por um cliente e um servidor que trocam informações. Após a conexão ser estabelecida entre o servidor e o cliente, o servidor aguarda a recepção de informações do cliente. Essas informações são processadas no servidor, e respostas são enviadas de volta ao cliente, desde que o cliente siga as regras definidas pelo protocolo implementado.

Do ponto de vista mais técnico, no código do cliente, o processo começa com a criação de um socket por meio da função `socket()`, seguida pelo estabelecimento da conexão com a função `connect()`. Em seguida, um loop `while` é utilizado para manter a comunicação com o servidor até que a conexão seja encerrada, geralmente de acordo com regras definidas no protocolo. Dentro desse loop, as regras do protocolo são verificadas, e as funções `send()` são usadas para enviar mensagens ao servidor. Há também outro loop `while` que utiliza a função `recv()` para receber respostas do servidor em pacotes.

No código do servidor, o processo é semelhante, começando com a criação de um socket (`socket()`) e a aceitação de conexões entrantes (`connect()`). O servidor também utiliza um loop `while` para verificar e processar as condições do protocolo com o auxílio da função `send()`. Outro loop `while` com a função `recv()` é utilizado para receber dados em pacotes do cliente.

Essa arquitetura de cliente e servidor é fundamental para a troca de informações em redes de computadores e é comumente usada em uma variedade de aplicativos, desde comunicações de rede simples até sistemas web complexos. Através do uso de protocolos definidos, é possível estabelecer uma comunicação eficaz e confiável entre o cliente e o servidor.

Servidor:

No intuito de facilitar como desenvolvi meu código do cliente irei explicar etapa por etapa de uma forma genérica e objetiva . Em síntese , meu cliente suporta várias operações, como instalar sensores, remover sensores, atualizar informações de sensores, solicitar informações de sensores e listar todas as informações de sensores disponíveis.

1. Função usage() é chamada para exibir uma mensagem de uso quando os argumentos da linha de comando não são fornecidos corretamente.

2. A função main inicia a execução do programa.

3. O código inicia a criação de um soquete (socket) e configura opções para reutilização de endereço (SO_REUSEADDR). Em seguida, associa (bind) o soquete a um endereço específico. Isso permite que o servidor escute conexões nesse endereço e porta específicos, preparando-se para aceitar conexões de clientes.

4. O código inicializa várias variáveis, como **valor1**, **valor2**, **valor3**, **valor4**, **potencia**, **linhas**, **colunas** e uma matriz chamada **matriz**. Essas variáveis são usadas para armazenar informações dos sensores e essa matriz é usada para armazenar informações sobre sensores, como o ID do sensor, potência e outros detalhes.

5. O código entra em um loop infinito, aguardando conexões de clientes e quando uma conexão é aceita, o programa entra em um segundo loop que lida com as operações do cliente

6. O código lê dados do cliente usando **recv** e processa as operações solicitadas pelo cliente.Essa operações suportadas incluem instalação de sensores, remoção de sensores, atualização de informações de sensores, solicitação de informações de sensores e listagem de informações de sensores.Após essas operações o código realiza verificações para garantir que as operações sejam realizadas corretamente e envia respostas ao cliente com mensagens apropriadas, como "sensor already exists" ou "successful installation".

7. Após a comunicação com o cliente, o código fecha o soquete do cliente e aguarda por novas conexões.

Cliente:

1. A função principal **main** é onde o programa começa sua execução.

2. Ela começa verificando se pelo menos dois argumentos de linha de comando (endereço IP do servidor e porta do servidor) foram fornecidos. Se não, o programa chama a função **usage** para mostrar ao usuário como usar o programa e, em seguida, encerra.

3. A função **addrparse** é chamada com os argumentos **argv[1]** (endereço IP do servidor) e **argv[2]** (porta do servidor) e armazena as informações do servidor em uma estrutura de dados **sockaddr_storage**.

4. Em seguida, o programa cria um soquete (**socket**) usando as informações do servidor, configurando-o como um soquete de fluxo (**SOCK_STREAM**).

5. Depois, ele faz uma conexão (**connect**) com o servidor usando o soquete e a estrutura de dados do servidor.

6. Após estabelecer a conexão, o código exibe uma mensagem informando que a conexão foi estabelecida com sucesso. Em seguida, ele entra em um loop infinito que permite ao usuário interagir com o programa.

7. O programa lê mensagens do usuário a partir do terminal (usando **fgets**) e realiza uma série de verificações e ações com base no conteúdo das mensagens.

8. As verificações e ações incluem a manipulação de mensagens relacionadas a comandos, como "install param", "install file", "change param", "change file", "show value", "show values", "remove", e "kill". O código interpreta essas mensagens, verifica se os valores fornecidos são válidos e, em seguida, envia solicitações ao servidor correspondentes aos comandos.

9. Para as mensagens relacionadas a comandos, o código formata as solicitações apropriadas, como "INS_REQ" para instalação de parâmetros e "CH_REQ" para alteração de parâmetros, e as envia para o servidor. Ele também lida com mensagens de resposta do servidor e as exibe no terminal, como dados de sensores.

10. O loop principal continua até que o usuário envie um comando "kill" ou uma mensagem de saída inválida (que faz com que **invalidsensor** seja definido como 1).

11. O código manipula as mensagens recebidas do servidor, exibindo informações sobre sensores e outros dados relevantes no terminal.

Discussão:

Os códigos do cliente e do servidor demonstram a importância de ter um protocolo de comunicação bem definido para permitir que as partes se entendam. Além disso, ambos os códigos mostram como lidar com erros e situações inesperadas. O cliente verifica os comandos e os valores fornecidos pelo usuário para evitar comandos inválidos ou valores de parâmetros incorretos. O servidor, por sua vez, pode responder com mensagens de erro quando as solicitações não podem ser atendidas.

No entanto, é importante ressaltar que, apesar da comunicação entre cliente e servidor estar funcionando corretamente, questões de segurança, escalabilidade e gerenciamento de recursos não foram implementadas. Esses aspectos são fundamentais em um projeto de redes de computadores e devem ser considerados para garantir o bom funcionamento e a integridade do sistema em ambientes reais. Portanto, futuros desenvolvimentos ou implementações mais robustas devem abordar essas preocupações para tornar o sistema mais completo e seguro.

Conclusão:

Em síntese, conclui-se que o cliente permite que o usuário interaja com o servidor por meio de comandos de terminal, enquanto o servidor responde a esses comandos executando ações correspondentes de acordo com o que foi especificado na implementação. Sendo assim, este trabalho ilustra um exemplo prático de comunicação em rede e comunicação cliente-servidor, demonstrando como um protocolo pode ser usado para definir o formato das mensagens trocadas entre ambas as partes.