Servidor de Multiplas Conexões

Trabalho Prático Individual

Valor: 25 pontos

Data de entrega: 21 de Novembro de 2022, segunda-feira, até às 23:59.

Introdução

Uma fábrica de equipamentos deseja dar um passo importante para revolucionar seu processo de produção de equipamentos industriais: utilizar equipamentos autônomos. Uma gerente realizou a compra destes equipamentos que estão prontos para serem instalados, mas estes modernos aparelhos precisam se comunicar uns com os outros. Assim, surge a necessidade de uma infraestrutura robusta de comunicação capaz de suportar conexões simultâneas entre equipamentos como mostra a Figura 1.

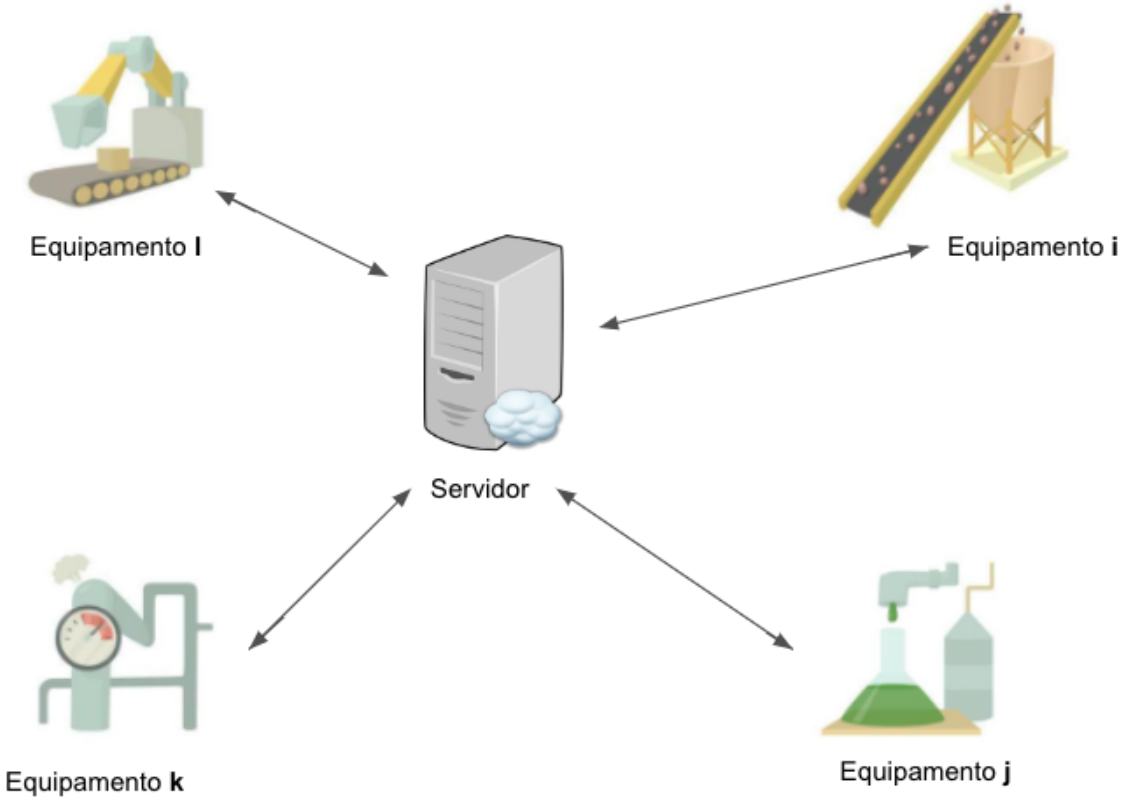

Figura 1: Exemplo de indústria Inteligente com equipamentos autônomos

Neste trabalho você irá desenvolver um protocolo que suporta as múltiplas conexões entre os equipamentos e a comunicação entre eles. Você deve desenvolver um **servidor**, responsável por coordenar as múltiplas conexões a medida que os equipamentos são conectados e desconectados da rede. Você também deve desenvolver os **clientes** (equipamentos) que devem ser capazes de solicitar informações uns dos outros por intermédio do servidor.

Toda conexão deve utilizar a interface de sockets da linguagem C. O sistema desenvolvido deve permitir a troca de mensagens de texto utilizando funcionalidades da biblioteca POSIX e comunicação via protocolo TCP. O servidor

deve ser *multi-threaded,* as múltiplas *threads* serão utilizadas para a manutenção das múltiplas conexões.

A próxima seção apresenta detalhes do formato de cada uma das mensagens, e como os equipamentos interagem com a rede.

Protocolo

O protocolo utilizado para troca de mensagens entre o cliente e o servidor controlador deve ser construído sobre o protocolo TCP. Cada mensagem deve comportar um texto contendo somente letras, números e espaços.

Esta seção apresenta as mensagens utilizadas na comunicação de controle e dados da rede, bem como as mensagens de erro e confirmação. Nas tabelas abaixo, as células com valor "–" correspondem a campos que não precisam ser definidos nas mensagens. A colula "Tipo" também não precisa ser adicionada à mensagem, para facilitar a implementação, utilize apenas a identificação da mensagem (ID msg).

Mensagens de Controle

Tipo	**ID Msg**	**Id Origem**	**Id Destino**	**Payload**	**Descrição**
REQ_ADD	01	-	-	-	Mensagem de requisição de conexão (entrada de equipamento na rede)
REQ_RM	02	IdEQ i	-	-	Mensagem de requisição de desconexão (remoção de equipamento na rede). IdEQi corresponde ao Id do equipamento solicitante
RES_ADD	03	-	-	IdEQ i	Mensagem de resposta contendo o Id do equipamento EQi
RES_LIST	04	-	-	IdEQ j, IdEQ k	Mensagem com lista de Ids de equipamentos. Onde IdEQ j e IdEQ k correspondem aos Ids dos equipamentos transmitidos na mensagem.

Mensagens de Dados

Tipo	**ID Msg**	**Id Origem**	**Id Destino**	**Payload**	**Descrição**
REQ_INF	05	IdEQ i	IdEQ j	-	Menagem de requisição de informação de um equipamento EQ i de origem para um equipamento EQ j de destino. Os campos EQ i e EQ j correspondem a Ids de equipamentos de origem e destino, respectivamente.
RES_INF	06	IdEQ j	IdEQ i	Dados (temperatura)	Mensagem de resposta de informação de um equipamento EQ j de origem para um equipamento EQ i de destino. Os campos EQ j e EQ i correspondem a Ids de equipamentos de origem e destino, respectivamente. O campo payload corresponde ao valor do dado (temperatura) do equipamento de Id EQ i.

Mensagens de Erro e Confirmação

Tipo	ID Msg	Id origem	Id Destino	Payload	Descrição
ERROR	07	-	IdEQ j	Code	Mensagem de erro enviada do servidor para o equipamento de Id EQ j. O payload contém o código de erro. Descrição de cada código possível: 01: "Equipment not found" 02: "Source equipment not found" 03: "Target equipment not found" 04: "Equipment limit exceeded"
OK	08	-	IdEQ j	Code	Mensagem de confirmação enviada do servidor para o equipamento de Id EQ j. O campo payload contém o código de confirmação. O único código possível:

					01: "Success"

Fluxo de Mensagens de Controle

Esta seção descreve as mensagens de **controle** transmitidas entre os equipamentos e o servidor afim de controlar a comunicação dos equipamentos na rede. Também são especificadas as decisões e mensagens que devem ser exibidas no terminal pelos equipamentos e servidor.

Abertura de Conexão com Servidor

1. Um equipamento **EQ i,** solicita a conexão ao servidor afim de obter um identificador na rece. Essa solicitação é feita através do envio da mensagem **REQ_ADD.**
2. O servidor recebe a mensagem de **EQ i** e verifica se a quantidade máxima de conexões permitida foi alcançada.
    a. Em caso positivo, o servidor responde a mensagem de erro **ERROR(04)** para o equipamento **EQ i**.
        i. O equipamento **EQ i** recebe a mensagem **ERROR(04)** e imprime no terminal a mensagem de erro com código correspondente (veja a tabela de mensagens de erro).
    b. Em caso negativo, o servidor define um identificador **IdEQ i** para **EQ i**, registra **IdEQ i** em sua base de dados e imprime no terminal a mensagem "`Equipment <IdEQ_i> added`", e envia a todos os equipamentos conectados na rede (*broadcast*) o Id do novo equipamento por meio da mensagem **RES_ADD(IdEQ i)**.
        i. Todos os equipamentos atualmente conectados na rede recebem a mensagem **RES_ADD(IdEQ i)**, registram o Id do equipamento **EQ i** em sua base de dados, e imprimem no terminal a mensagem "`Equipment <IdEQ_i> added`". O equipamento **EQ i** por sua vez, ao receber a mensagem **RED_ADD(IdEQ i)** registra sua nova identificação e imprime no terminal "`New ID: <IdEQ_i>`"
        ii. O servidor envia para o **EQ i** (*unicast*) a lista de equipamentos que estão atualmente conectados na rede através da mensagem **RES_LIST(IdEQ j, IdEQ k).**
        iii. O equipamento EQ i recebe a lista de equipamentos pela mensagem **RES_LIST(IdEQ j, IdEQ k)** e registra os novos equipamentos na sua base de dados.

Encerramento de Conexão com Servidor

1. Um equipamento **EQ i** solicita ao servidor o encerramento da sua conexão com a rede através da mensagem **REQ_RM(IdEQ i)**.
2. O servidor recebe a mensagem **REQ_RM(IdEQ i)** e verifica se **IdEQ i** existe na base de dados.
    a. Em caso negativo, o servidor responde ao equipamento **EQ i** a mensagem de erro **ERROR(01)**.
        i. O equipemtno **EQ i** recebe a mensagem **ERROR(01)** e imprime no terminal a descrição correspondente a mensagem (veja a tabela de mensagens de erro).
    b. Em caso positivo, o servidor remove o equipamento **EQ i** da base de dados, e responde a mensagem **OK(01)** ao equipamento **EQ i**. O servidor também imprime no terminal `"Equipment IdEQi removed"` e envia a mensagem **REQ_REM(IdEQ i)** para todos equipamentos atualmente conectados na rede (broadcast).
        i. O equipamento **EQ i** recebe a mensagem **OK(01)**, imprime no terminal a descrição da mensagem, fecha sua conexão com o servidor e encerra sua execução.
        ii. Todos os equipamentos atualmente conectados a rede recebem a mensagem **REQ_REM(IdEQ i),** removem o equipamento IdEQ i da sua base de dados, e imprimem no terminal `"Equipment IdEQ i removed"`.

Fluxo de Mensagens de Dados

Esta seção descreve as mensagens de **dados** transmitidas entre os equipamentos e o servidor afim de trocarem informações uns com os outros. Também são especificadas as decisões e mensagens que devem ser exibidas no terminal pelos equipamentos e servidor.

Equipamento EQ i solicita informações de outro equipamento EQ j

1. Um equipamento **EQ i** solicita ao servidor informações do equipamento **EQ j** por meio da mensagem **REQ_INF(IdEQ i, IdEQ j)**.
2. O servidor recebe a mensagem **REQ_INF(IdEQ i, IdEQ j)** e verifica a existencia do equipamento **IdEQ i** em sua base de dados.
    a. Em caso negativo, o servidor imprime no terminal a mensagem `"Equipment IdEQ i not found"` e responde ao **EQ i** a mensagem de erro **ERROR(02)**.

i. O equipamento **EQ i** recebe a mensagem **ERROR(02)** do servidor e imprime no terminal sua descrição, conforme da tabela.

b. Em caso positivo o servidor verifica a existencia do equipamento **EQ j** em sua base de dados
   i. Caso negativo, o servidor imprime no terminal "`Equipment IdEQ j not found`" e responde a mensagem de erro **ERROR(03)** para o equipamento **EQ i**.
      1. O equipamento **EQ i** recebe a mensagem **ERROR(04)** do servidor e imprime no terminal sua descrição, conforme da tabela.
   ii. Caso positivo, o servidor envia a mensagem **REQ_INF(IdEQ i, IdEQ j)** ao equipamento **EQ j**.
      1. Equipamento **EQ j** recebe mensagem **REQ_INF(IdEQ i , IdEQ j)**, imprime no terminal "`requested information`", gera informação (valor aleatório) e responde ao servidor as informações por meio da mensagem **RES_INF(IdEQ j IdEQ i , payload)**.
      2. Servidor recebe a mensagem **RES_INF(IdEQ j IdEQ i , payload)** e verifica se o equipamento **IdEQ j** existe na base de dados.
         a. Em caso negativo, o servidor imprime no terminal "`Equipment IdEQ j not found`" e responde ao **EQ j** a mensagem **ERROR(02).**
            i. O equipamento **EQ j** recebe a mensagem **ERROR(03)** e imprime no terminal sua descrição conforme a tabela.
         b. Em caso positivo, o servidor verifica a existencia do eqiupamento EQ i na base de dados.
            i. Em caso negativo, Servidor imprime em tela "`Equipment IdEQ i not found`" e responde mensagem de erro **ERROR(03)** para equipamento **EQ i**.
               1. O equipamento **EQ i** recebe **ERROR(03)** e imprime no terminal sua descrição.
            ii. Em caso positivo, o servidor repassa a mensagem **RES_INF(IdEQ j, IdEQ i, PAYLOAD)** para **IdEQ i.**
               1. IdEQ i recebe a mensagem **RES_INF(IdEQ j, IdEQ i, PAYLOAD)** e imprime no terminal "`Value from IdEQ j : <payload>`".

* O valor aleatório que representa a temperatura do equipamento deve ser um número decimal aleatório entre 0 e 10 com 2 casas decimais (ex. 2.13, 5.12 ou 10.94).

Implementação

Tanto o cliente (equipamentos) quanto o servidor devem ser implementados com POSIX sockets na **linguagem C**, utilizando apenas a **biblioteca padrão de sockets** TCP. Seu programa deve ser executável no sistema operacional **Linux**, não precisa funcionar em Windows.

Haverá apenas um socket em cada equipamento (cliente), independente de quantos outros programas se comunicarem com aquele processo. Utilize as funções *send* e *recv* para enviar e receber mensagens. No caso do servidor, ele deve manter um socket para receber novas conexões (sobre o qual ele executará a função *accept*) e um socket para cada cliente conectado.

O servidor deve suportar conexões do tipo **IPv4**.

O servidor deve tratar até **10 conexões simultâneas**. O servidor é também responsável por **identificadores únicos** para cada equipamento (cliente). Cada equipamento (cliente) é iniciado sem identificação, ele recebe sua identificador após realizar com sucesso sua solicitação de conexão à rede através do servidor.

Tanto o servidor quanto os clientes (equipamentos) recebem entradas do teclado, e devem imprimir as mensagens recebidas no terminal.

Comandos

O equipamento deve responder a alguns comandos pela entrada do teclado:

- **`close connection`**: este comando requisita o encerramento da conexão do equipamento com a rede. Quando enviado ao servidor, deve disparar o fluxo de mensagem de controle "Encerramento de conexão com servidor".
- **`list equipment`:** este comando lista os equipamentos existentes na base de dados do equipamento em questão, por Id. Exemplo de resposta: `"<IdEQ_i> <IdEQ_j> <IdEQ_k>"`

- **`request information from <IdEQ_j>:`** comando requisita informações de temperatura do equipamento EQ j, dispara o fluxo de mensagem de dados "Equipamento EQ i solicita informações de outro equipamento EQ j"

Execução

O servidor deve receber, o número de porta na linha de comando especificando em qual porta ele receberá conexões. O equipamento (cliente) deve receber, **necessáriamente nesta ordem**, o endereço IP e a porta de onde estabelecerá conexão. Exemplo de dois programas executando em terminais diferentes:

servidor terminal 1 > ./server 51511
cliente terminal 2 > ./equipment **127.0.0.1** 51511
cliente terminal 3 > ./equipment **127.0.0.1** 51511

Dicas e Material para consulta

A playlist de Introdução a Programação em Redes mostra o passo a passo do desenvolvimento de algumas aplicações, com exemplos de troca de mensagens com socket TCP.

O guia do Beej para Programação em Redes é um dos mais completos e didáticos disponíveis na internet. Apresenta em detalhes como funcionam e quando podem ser utilizadas as funções da biblioteca de sockets da linguagem C. Contém muitos exemplos.

O capítulo 6 do livro “TCP IP Sockets in C, Second Edition Practical Guide for Programmers” disponível do moodle explica como lidar com multi-tasking em cenários multi-sockets e multi-threads.

Pode ser uma boa ideia implementar este trabalho por etapas: primeiro o tratamento de multiplas conexões, depois criar o formato das mensagens, e por fim as mensagens dos equipamentos e servidor.

Entrega

Cada aluno deve entregar documentação em formado PDF contendo no **máximo 06 páginas,** sem capa e fonte tamanho 12. Seu relatório deve conter a descrição da arquitetura utilizada pelo servidor, os refinamentos dos cenários implementados, as estruturas de dados utilizadas, e suas decisões de implementação que não estão listadas neste documento.

Como sugestão, considere incluir as seguintes seções no relatório: **introdução**, **arquitetura**, **servidor**, **equipamento**, **discussão** e **conclusão**. A documentação corresponde a 20% da pontuação do trabalho (5 pontos), portanto, dedique tempo à desenvolvê-la bem.

Cada aluno deve entregar também o código fonte em C, e um Makefile para compilar o programa.

- O Makefile deve compilar o cliente em um binário chamado “equipment” e o servidor em um binário chamado “server”
- Seu programa deve ser compilado ao se executar apenas o comando “make”, ou seja, sem a necessidade de parâmetros adicionais.
- Os nomes dos arquivos devem ser padronizados: `server.c`, `equipment.c` e `common.c` ou `common.h` (se necessário utilizar arquivo com funções comuns).
- A entrega deve ser feita no formato ZIP, com o nome seguindo o padrão: TP2_MATRICULA.zip

Importante: Teste seu código **antes** de submeter a tarefa no moodle. Corrija eventuais erros de compilação e tenha certeza de que tudo funcione corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cópia de trabalhos.

Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.