Trabalho prático 1 - Redes de Computadores TE

Engenharia de Sistemas 2024/01 - Professor Aldri Luiz dos Santos

Aluna: Maria Clara Oliveira Domingos Ruas

Matrícula: 2021019572

1. Introdução

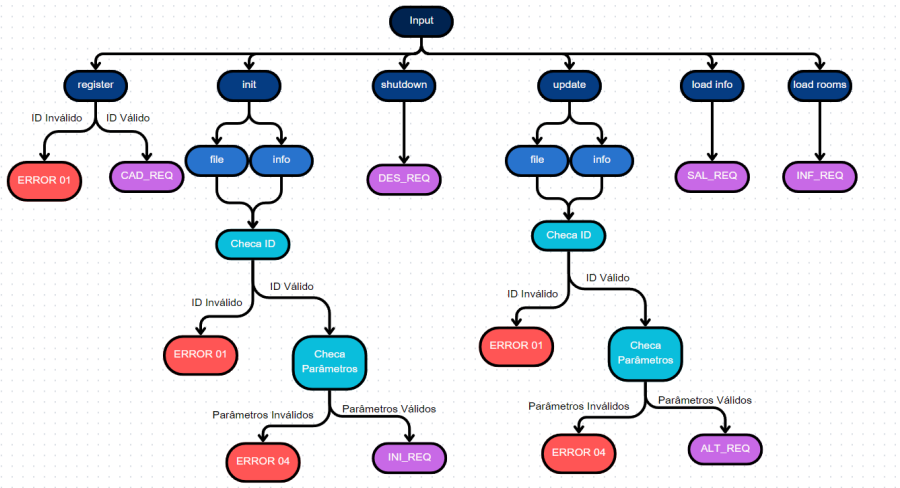
Tendo em mente as novas condições climáticas e o conforto dos alunos, a proposta de monitoramento das condições das salas de aula foi requisitada. Na solução, foi realizado um protocolo de comunicação TCP/IP para o monitoramento dos ambientes de aprendizado. O TCP é um protocolo de comunicação em redes que garante a entrega das informações, sendo mais confiável quando comparado com o UDP.

A solução foi implementada em C, utilizando da biblioteca Socket para realizar a comunicação entre as duas partes da rede.

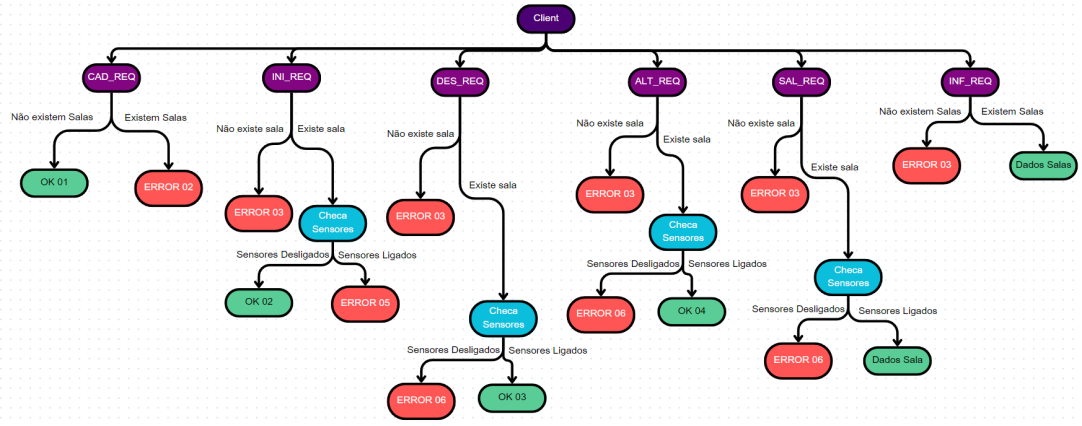
2. Arquitetura

Na solução, existe o Server, que armazena e organiza todas as informações durante o período de uso, e o Client, que coleta os dados, faz um tratamento inicial, e envia os pedidos para o Server. A comunicação entre os 2 hosts é feita por meio dos diferentes códigos definidos para o trabalho.

A primeira parte é o tratamento de comando no Client. Aqui, um comando é pego inicialmente pelo terminal, e o fluxo inicial de como deve ser tratado os comandos é:



Os comandos em vermelho são os erros identificados logo na passagem de parâmetros, e em roxo, os códigos que são enviados para o servidor. O próximo fluxo mostrará como os códigos enviados do cliente são tratados.



Depois que o servidor envia uma mensagem de resposta, novamente no cliente é feito um tratamento para cada tipo de mensagem de Error ou de confirmação, onde é basicamente impresso no terminal a descrição do erro ou da confirmação. O Server e o Client são dois arquivos independentes que se comunicam pela rede. Assim, mesmo passando por cima brevemente das funcionalidades, agora iremos detalhar como cada arquivo foi implementado.

3. Servidor

No servidor, foi criado um struct classroom para armazenar todas as informações de uma sala. Logo após, é criado um array de classroom, que servirá como base de dados para todas as salas de aula que serão guardadas. O array tem tamanho 8, já que é possível cadastrar no máximo 8 salas (ID's variam de 0 a 7).

```
/*
    Descricao: Struct de uma sala
    Valores (int): ID da Classe, valores dos sensores(Temperatura, Umidade
    Status dos ventiladores 1, 2, 3 e 4)
*/
struct classroom {
  int classID;
  int temp, umidade, ven1, ven2, ven3, ven4;
}classroom;

//Array de salas, no máximo 8 salas (ID's podem variar de 0 a 7)
struct classroom classes[8];
```

Movendo para as funções auxiliares:

compareComand: Compara se duas strings são exatamente iguais.

pegaParam: Baseado em uma string de input, coleta todos os dados de uma sala (ID da sala, temperatura, umidade e status dos ventiladores 1, 2, 3 e 4). As entradas são em ponteiros, então já são atualizadas diretamente no array das salas.

trataCod: Identifica qual foi o código enviado pelo cliente, e retorna o número interno para o tratamento.

Movendo para o main, ele é dividido em 3 etapas:

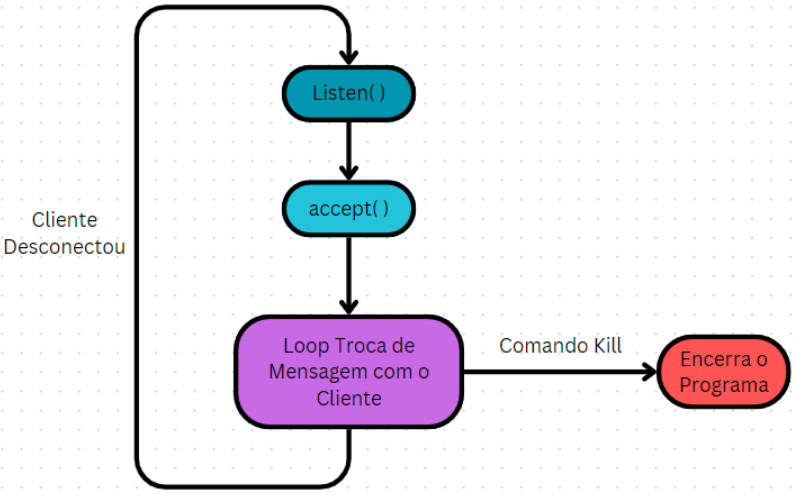
- Criação do socket (IPV4 ou IPV6)
- Loop de Conexão com o cliente
- Loop de Mensagem do cliente conectado

Criação do Socket:

A criação do socket varia dependendo da entrada argv. Se for v4, é criado um socket IPV4, e se for v6, é criado um socket IPV6.

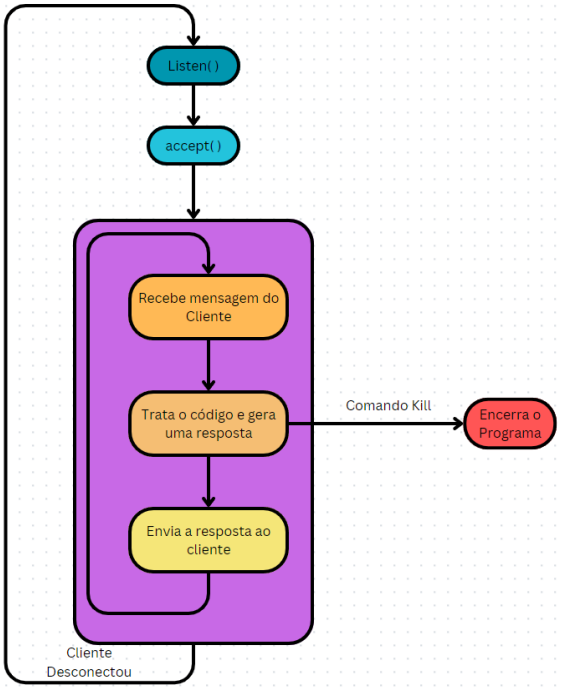
Loop de Conexão com o cliente:

No primeiro loop do servidor, ele garante que o servidor sempre estará aguardando por novas conexões, e quando conectar, entra no loop seguinte. Este loop só é encerrado quando o cliente envia um comando kill, que encerra o servidor.

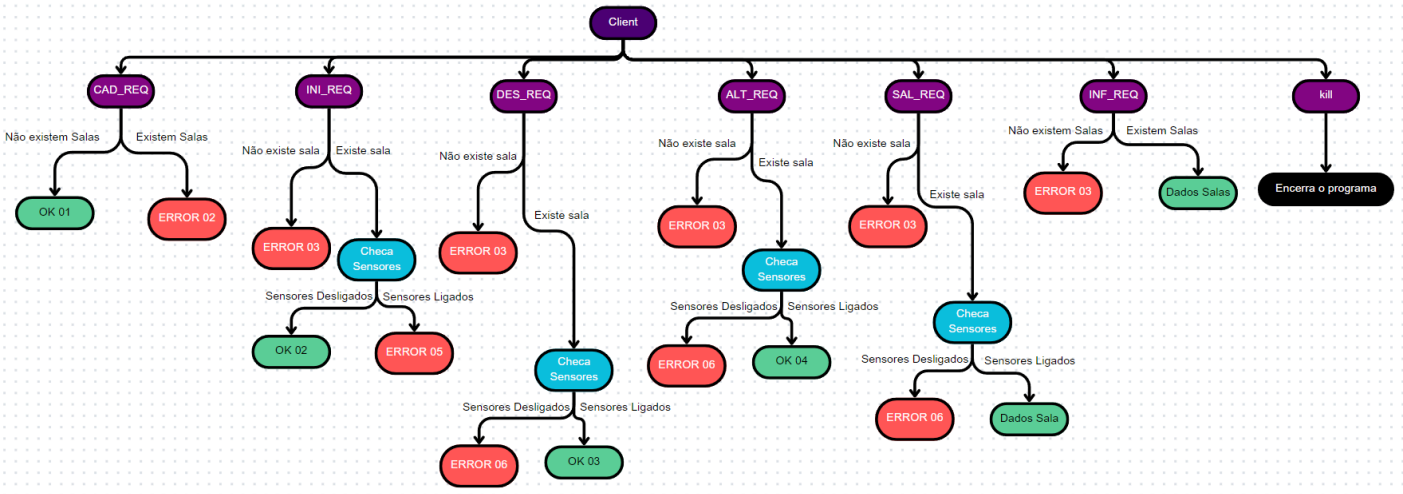


Loop de Mensagem do cliente conectado:

O loop interno é utilizado para responder as mensagens do cliente conectado enquanto ele estiver enviando mensagens. Para sair deste loop, o cliente é desconectado ou é identificado o comando kill.



O tratamento de código segue um fluxo semelhante ao apresentado na arquitetura, apenas adicionando o tratamento do comando kill.



4. Cliente

O cliente é responsável por toda a interação e exibição de mensagens. Nele, é feito um tratamento inicial dos comandos, o envio dos códigos para o servidor, e o recebimento da resposta e tratamento desta para a exibição de sua mensagem correspondente. As funções auxiliares utilizadas foram:

compareComand e pegaParam: Funcionalidades idênticas às funções de mesmo nome no servidor. Na pegaParam, também é checado se não existem parâmetros a mais do que esperado.

sendMessage: Envia o input char * message pelo caminho do socket fornecido.

checkError: Recebe uma string, correspondente à resposta do servidor, e exibe a mensagem referente ao código ou as informações da base de dados fornecidas pelo servidor.

checkParam: Verifica se todos os parâmetros estão dentro do que é válido (temperatura entre 0 e 40, umidade entre 0 e 80, status dos ventiladores na ordem e com valores de status entre 0 e 2, nenhum parâmetro faltando).

tipoComando: Faz uma seleção inicial de qual é o comando que foi pego pelo terminal, retornando um código interno referente a cada comando.

Para as funções de tratamento, foi buscado reutilizar o máximo de código. Assim, as 6 funcionalidades requeridas foram divididas em 3 categorias: comandos de alteração que precisam apenas do ID (Registro e Shutdown), comandos de alteração que possuem todos os parâmetros (Init e Update) e comandos de busca de informação (Load info e Load rooms).

tratamentoRegistro: Tratamento dos comandos de register e shutdown. Pega o ID da sala (no registro, checa se esse ID é válido), também confere se não existe mais inputs do que o esperado. Após as verificações, é montado a mensagem para ser enviada ao servidor e colocado na variável global cod.

tratamentoInit: Tratamento dos comandos de init e update. Usando pegaParam e checkParam, coleta todos os parâmetros relacionados à sala e checa se estes são válidos. Após a verificação, é montado a mensagem para ser enviada ao servidor e colocado na variável global cod.

tratamentoLoad: Tratamento dos comandos load rooms e load info. Para load rooms, apenas é montado a mensagem de envio para o servidor e armazenado em cod. Para load info, o ID da sala é coletado, e a mensagem de envio é armazenada em cod.

Movendo para o main, ele é dividido em 2 etapas:

- Criação do socket (IPV4 ou IPV6) e conexão com o servidor
- Loop de input do terminal

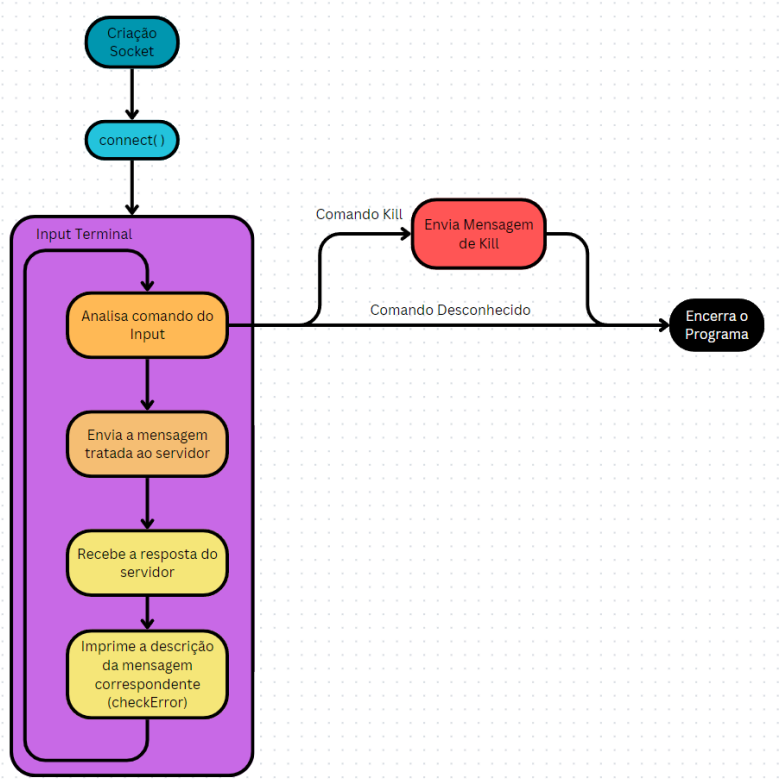
Criação do Socket:

A criação do socket varia dependendo da entrada argv. Na execução, é colocado o endereço IP do servidor e a porta. A partir do endereço e da função ipton, é identificado se é necessário a criação do socket IPV4 ou IPV6. Após a criação, é enviado um pedido de conexão para o servidor. Se não tiver nenhum erro na conexão, o programa continua para a próxima etapa.

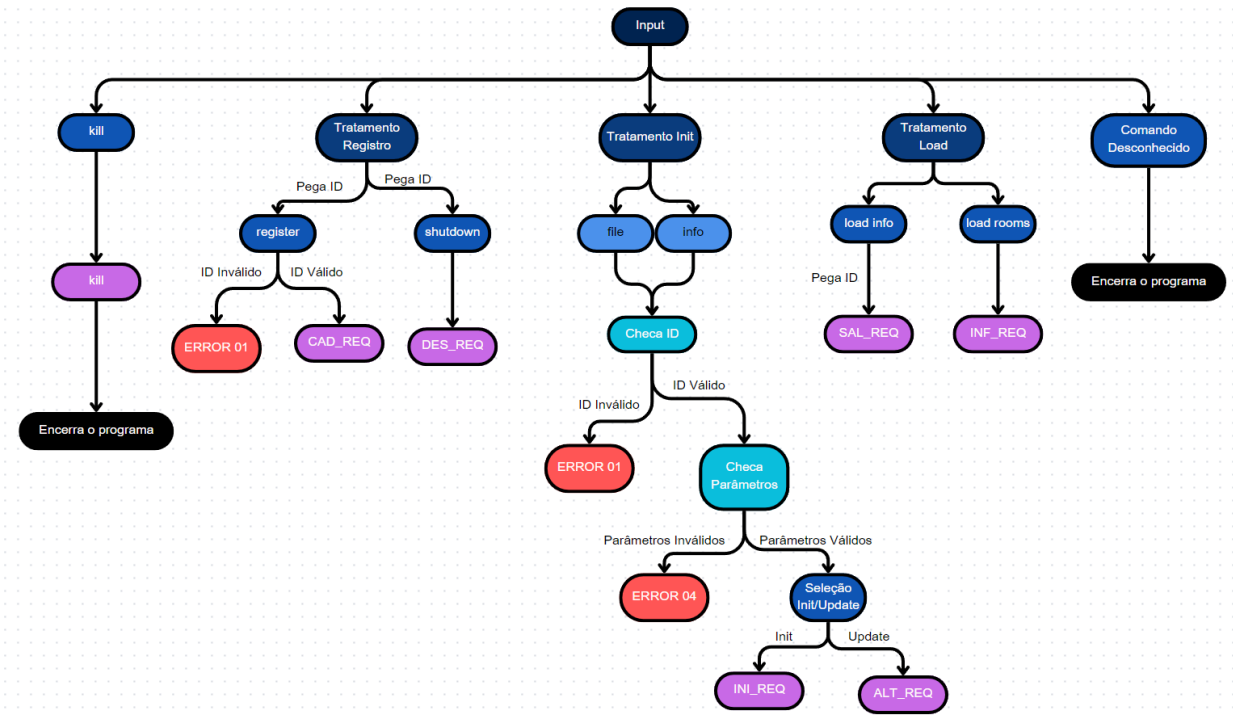
Loop de input do terminal:

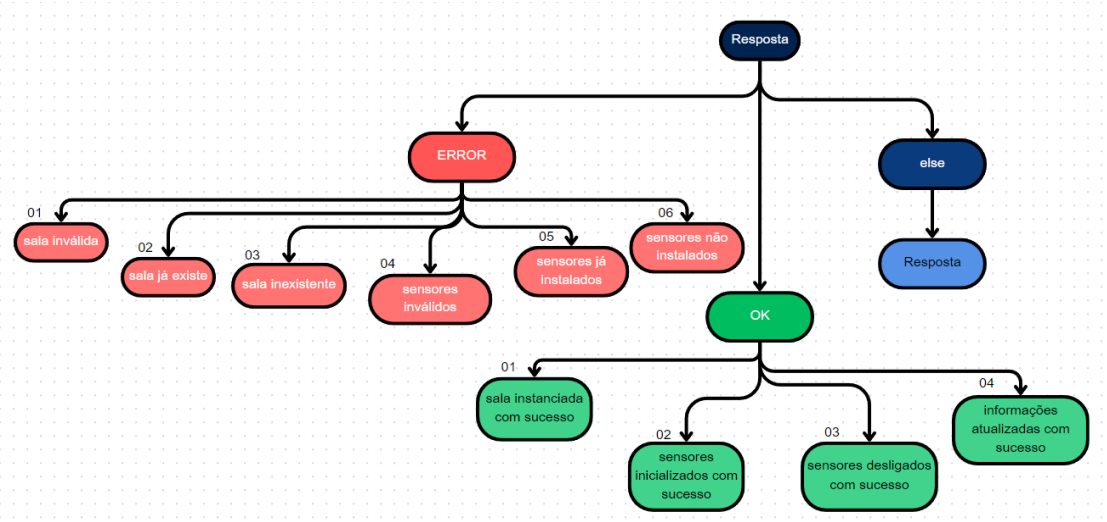
Aqui, enquanto estiver recebendo input's do terminal, o loop continuará rodando, armazenando cada linha em uma variável temporária. Para sair do loop e encerrar o programa, existem duas formas: pelo comando kill, que encerra tanto o cliente quanto o servidor, ou quando é digitado um comando desconhecido, neste caso, apenas o cliente se encerra e o servidor continua funcionando.

Para cada linha pega pelo terminal, é analisado qual comando foi digitado. Identificado o comando, prossegue o tratamento, coletando todas as informações e parâmetros necessários utilizando as funções de tratamento. Se erros são identificados, já é tratado diretamente no cliente. Se nenhum erro foi identificado a princípio, no final do tratamento, é enviado uma mensagem para o servidor, e logo após, o cliente recebe a mensagem de resposta do servidor. A mensagem é tratada e é impresso sua descrição, e o cliente volta a esperar um novo comando.



O tratamento de código segue um fluxo semelhante ao apresentado na arquitetura, buscando otimizar algumas estruturas, reutilizar funções, e adicionando o tratamento de kill e comando desconhecido. Posteriormente, o fluxo também do tratamento de resposta, onde no fim é a mensagem impressa no terminal.





5. Discussão

Durante o desenvolvimento do código, houveram problemas relacionados com a conexão do socket IPV6 e com o recebimento das respostas às mensagens por parte do servidor. Posteriormente, estes erros foram resolvidos, e o trabalho seguiu sem maiores problemas.
Pensando em uma maior reutilização de código, funções iguais tanto no servidor quanto no cliente poderiam ter sido alocadas em um arquivo comum externo, mas pensando no cenário de um cliente e um servidor completamente independentes, optei por fazer apenas os dois arquivos principais. No mais, todas as funções de monitoramento das salas de aula conseguiram ser atendidas.

6. Conclusão

Com o trabalho, foi possível observar o funcionamento de uma comunicação em rede utilizando o protocolo TCP/IP. O monitoramento das salas de aula utilizou tanto o servidor para armazenar as informações quanto o cliente para uma interface de comunicação com o usuário (inputs por meio do CMD).

É interessante observar na prática os conceitos estudados em sala de aula, como o three-handshake para a conexão de duas pontas, o fechamento de conexão, e os padrões da internet para endereçamento de máquinas. O fluxo temporal de envio e recebimento de mensagens também pôde ser exemplificado com a comunicação entre o cliente e o servidor, com um protocolo que foi construído a partir das orientações do enunciado.

Neste trabalho, o servidor aceitava a conexão de apenas um cliente, e conseguia se comunicar com eficácia com este. Em trabalhos futuros, será interessante ver como é o funcionamento de um servidor com múltiplas conexões.