DOCUMENTAÇÃO – TRABALHO PRÁTICO 1 – REDES DE COMPUTADORES
VICTOR DELLA CROCE MALTEZ – 2019042392

1. Introdução

O código explica a seguir foi desenvolvido com objetivo de resolver o seguinte desafio: desenvolver um sistema em rede de controle de informações climáticas capaz de registrar e monitorar dados sobre as salas de aula a partir da criação de uma Unidade de Controle (UC), no qual há o fluxo de informações e controles no sistema (servidor), e uma Unidade de Monitoramento (UM) (cliente), o qual tem as seguintes funcionalidades: cadastro de sala de aula, coleta de dados de temperatura, umidade do ar e estados dos ventiladores através de sensores e retornar ao servidor por meio de protocolos.

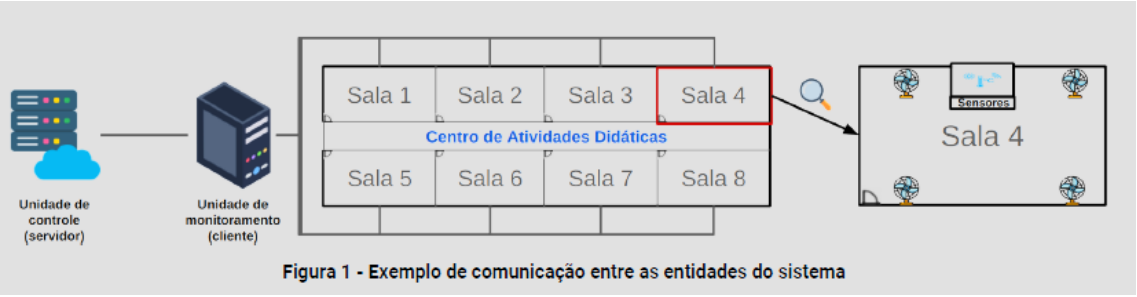

Figura 1 - Exemplo de comunicação entre as entidades do sistema

2. Arquitetura

O sistema é baseado em uma arquitetura cliente-servidor, onde o servidor gerencia as salas e os sensores, e o cliente interage com o servidor enviando solicitações e recebendo respostas. A comunicação entre cliente e servidor é feita por meio de sockets TCP/IP, garantindo a confiabilidade na transferência de dados.

O servidor é implementado em C e utiliza um loop principal para aceitar conexões de clientes. Ele armazena as informações das salas em um array de estruturas sala, onde cada estrutura representa uma sala com um ID único, dados dos sensores e estados dos ventiladores.

O cliente também é implementado em C e permite ao usuário interagir com o servidor enviando mensagens que indicam as operações desejadas, como criação de sala, inicialização de sensores, desligamento de sensores, atualização de valores dos sensores, solicitação de informações de uma sala específica e solicitação de informações de todas as salas. O cliente exibe as respostas do servidor para o usuário.

3. Cliente-Servidor

Para desenvolvimento da Unidade de Monitoramento (UM – Cliente) e da Unidade de Controle (UC – Servidor), foram utilizadas algumas funções e constantes em comum, as quais serão explicadas abaixo para facilitar compreensão e evitar repetição na documentação.

i. Funções

1. **addParse(const char *addrstr, const char *portstr, struct sockaddr_storage *storage)**
   Converte um endereço IP (IPv4 ou IPv6) e uma porta de string para uma estrutura sockaddr_storage que pode ser usada em funções de socket. Ela primeiro converte a porta para o formato de rede e então tenta interpretar o endereço

como IPv4 e IPv6, preenchendo a estrutura apropriada com os dados.

2. **addrtostr(const struct sockaddr *addr, char *str, size_t strsize)**
   Converte uma estrutura sockaddr para uma string IP legível. Ela verifica se o endereço é IPv4 ou IPv6 e usa inet_ntop para realizar a conversão.

3. **server_sockaddr_init(const char *proto, const char* portstr, struct sockaddr_storage *storage)**
   Inicializa uma estrutura sockaddr_storage para uso em um servidor, a qual recebe o protocolo ("v4" para IPv4 ou "v6" para IPv6), a porta como string, converte a porta para o formato de rede e preenche a estrutura com o endereço apropriado (INADDR_ANY para IPv4 e in6addr_any para IPv6) e a porta fornecida.

ii. Constantes

1. **BUFFSZ:** define o tamanho máximo (em bytes) das mensagens
2. **OKO1 (a 06):** definem, de acordo com o código de cada "OK", a respectiva mensagem que deve ser retornada.
3. **ERROR01 (a 06):** definem, de acordo com o código de cada erro, a respectiva mensagem que deve ser retornada
4. **MIN_ID_SALA/MAX_ID_SALA:** menor e maior, respectivamente, IDs válidos para as salas de aula [0-7].

b. Cliente

O cliente inicia criando um socket e se conectando ao servidor por meio de um endereço IP e porta específicos, permitindo ao usuário interagir com o servidor através de um menu simples, onde pode escolher a operação desejada. O cliente envia a solicitação correspondente ao servidor e exibe a resposta recebida.

Para desenvolvimento da Unidade de Monitoramento (Cliente) foram utilizadas algumas funções e constantes, as quais serão explicadas abaixo:

i. Funções

1. **valid_class_identifier(char *sala_id_s)**
   Verifica se um identificador de sala é válido. Ela recebe uma string contendo o identificador da sala e converte-o para um número inteiro, verificando se está dentro dos limites aceitáveis.

2. **get_datas_from_command_line(char *datas)**
   Extrai os valores dos dados dos sensores a partir de uma string de entrada fornecida na linha de comando. O retorno é uma nova string contendo apenas os valores dos sensores.

3. **get_datas_from_file(char *filename)**
   Lê os dados dos sensores de um arquivo e retorna uma string contendo esses valores. Ela verifica se o arquivo foi

aberto corretamente e aloca memória para armazenar os dados lidos.

4. **valid_sensor_values_identifier(char *str)**
   Verifica se os valores dos sensores e ventiladores são válidos. Ela recebe uma string contendo os valores dos sensores e ventiladores e verifica se estão dentro dos limites aceitáveis.

ii. Constantes

1. **MIN_TEMP_SENSOR/MAX_TEMP_SENSOR:** menor e maior, respectivamente, valor de temperaturas válido [0-40].

2. **MIN_UM_SENSOR/MAX_UM_SENSOR:** menor e maior, respectivamente, valor de umidade válido [0-100].

3. **MIN_ESTADO_VENT/MAX_ESTADO_VENT:** menor e maior, respectivamente, ID válido de estados dos ventiladores [0-2] (0 – com defeito, 1 – desligado, 2 – ligado).

4. **MIN_VALID_SENSOR_DATAS_SZ/MAX_VALID_SENSOR_DATAS_SZ:** menor e maior, respectivamente, tamanho das entradas com valores para os sensores [15-17] (os limites foram definidos a partir da análise do tamanho da menor e da maior entrada válida).

c. Servidor

O servidor inicia criando um socket e fazendo o 'bind' para um endereço IP e porta específicos. Em seguida, entra em um estado de espera (listen) para aceitar conexões de clientes. Para cada conexão aceita, o servidor cria um novo socket para se comunicar com o cliente.

O servidor processa as solicitações dos clientes em um loop, lendo as mensagens dos clientes, interpretando-as e enviando as respostas adequadas. Ele utiliza diferentes funcionalidades para cada tipo de solicitação, como: criar uma nova sala, inicializar sensores, desligar sensores, atualizar valores dos sensores e fornecer informações sobre as salas.

Para desenvolvimento da Unidade de Controle (Servidor) foram utilizadas algumas funções, as quais serão explicadas abaixo:

i. Funções

1. **sala* init_salas()**
   Inicializa um array de salas, alocando memória para 8 salas e atribuindo valores iniciais a seus campos. O retorno é um ponteiro para o array de salas inicializado.

2. **char* concatenar_salas(sala *salas)**
   Concatena informações das salas em uma única string, formatando-as de acordo com um padrão específico. O argumento passado (sala* salas) é um array

de salas contendo as informações a serem concatenadas. O retorno é uma string contendo as informações concatenadas das salas.

3. **command_identifier(char* buf, int _socket, int csock, sala *salas)**

   Identifica qual comando foi enviado pelo cliente ao servidor e, posteriormente, realiza os métodos que forem necessários. Os argumentos passados são a mensagem (buf), o socket do servidor (_socket) e do cliente (csock) e um vetor de (struct) salas, no qual armazena os dados das salas. O retorno é a mensagem que será enviada ao cliente (ou, no caso do 'kill', retorna "-1" como identificador de finalização do server).

4. Discussão

A simplicidade e diretividade do código são pontos positivos, facilitando a compreensão e o uso por parte dos usuários, com uma lógica de adição das salas e atualização de dados clara e fácil de entender, o que contribui para a facilidade de manutenção do código no futuro.

No geral, o código apresenta uma interação intuitiva para o usuário, deixando-o sempre a par dos resultados das execuções, sejam com resposta positivas (oks) ou negativas (erros), e permitindo que não tenha dificuldades em aprender e entender como realizar as ações desejadas.

5. Conclusão

O código atende ao seu propósito principal de permitir o cadastro de salas e atualização de dados, de forma simples e direta, com objetivo de ser um sistema com simplicidade e a clareza do código, facilitando sua compreensão e manutenção, o que é sempre positivo em termos de desenvolvimento de software.