Trabalho Prático 1 - Redes de Computadores

Vitor Hugo Lacerda Lana - 2018072301

22 de abril de 2024

1 Introdução

As recentes ondas de calor extremo, exacerbadas pelo aquecimento global, têm afetado negativamente o desempenho dos alunos em ambientes educacionais, especialmente em lugares como Belo Horizonte, Brasil, que registrou temperaturas significativamente acima da média em 2023. Diante desse desafio, a tecnologia de Internet das Coisas (IoT) surge como uma solução promissora, permitindo aprimorar a coleta e análise de dados ambientais e de desempenho estudantil em instituições de ensino. A implementação de dispositivos IoT em salas de aula possibilita o ajuste automático de iluminação e temperatura, promovendo um ambiente mais confortável e propício ao aprendizado.

Tendo como base esse contexto, esse documento tem como proposta trazer com mais detalhes de como foi realizada uma implementação de um programa piloto para resolução do problema acima. Será discutida a organização da arquitetura utilizada, entendendo melhor o funcionamento do cliente como unidade de monitoramento (UM) e sobre o servidor como unidade de controle (UC), os problemas encontrados durante o desenvolvimento do projeto e propostas de futuras melhorias.

2 Arquitetura

Para solução do problema proposto foi desenvolvido um utilizando a linguagem de programação C, para a parte de comunicação foi utilizada a biblioteca padrão de redes POSIX e para manipulação de strings a biblioteca string.h além das bibliotecas para entrada e saída de dados, seguindo o padrão do mercado de desenvolvimento de software a linguagem padrão para comentários e nomes das funções foi o inglês e como solicitado na especificações de desenvolvimento as mensagens impressas pelo sistema estão na língua portuguesa, caso necessário podem ser facilmente traduzidas para língua inglesa realizando a alteração de uma função no módulo *common.c*.

2.1 Organização dos módulos

O sistema desenvolvido foi separado em dois módulos, a unidade de monitoramento (UM) a qual foi implementada no arquivo *client.c* e a unidade de controle (UC) que foi implementada no aquivo *server.c*, em comum aos dois módulos temos o arquivo *common.c* onde foi feita a implementação das funções de manipulação dos comandos enviados e das respostas do servidor.

2.1.1 *common.c*

Antes de explicar as implementações do *client.c* e *server.c* precisamos explicar as implementações realizadas na *common.c*, foram criadas 28 funções para implementação completa do sistema proposto, todos possuem comentários no código anexo a esse documento, para melhor entendimento trarei as principais e explicarei o comportamento delas aqui, podemos separar essas funções em comportamentos macros como funções genéricos de sistema, manipulação de strings, manipulação de conexão de rede e funções específicas do projeto. Por questões de clareza da documentação não vão ser especificados os tipos de retorno e os parâmetros de entrada de cada função, mas ressalto que ambos estão detalhados com clareza no código anexo.

2.1.1.1 Estrutura de dados utilizada

Para gerenciamento das salas e sensores foi criada uma estrutura de dados que comportasse todos os dados necessários para implementação.

```
//Definição da estrutura para armazenar o estado de cada ventilador
typedef struct {
    int id;    // ID do ventilador
    int state; // Estado do ventilador (0 = Defeituoso, 1 = Desligado, 2 = Ligado)
} Fan;

// Definição da estrutura para armazenar as informações da sala de aula
typedef struct {
    int roomID;       // ID da sala
    int temperature; // Temperatura da sala em graus celcius
    int humidity;   // Humidade da sala em percentual
    Fan fans[4];      // Vetor de ventiladores sendor 4 ventiladores por sala
} Classroom;
```

2.1.1.2 Funções de manipulação de rede

setupTCPServerSocket()

A função é responsável por criar um socket de servidor TCP e vinculá-lo ao endereço local na porta definida pelo parâmetro fornecido. Implementada de maneira genérica, ela é compatível tanto com conexões TCP usando endereços IPv6 quanto IPv4.

setupTCPClientSocket()

A função é projetada para criar um socket de cliente TCP e estabelecer uma conexão com os endereços de IP e portas especificados nos parâmetros. Semelhante à função que cria um socket de servidor, esta também foi desenvolvida de maneira genérica, permitindo sua utilização tanto em conexões TCP com endereços IPv6 quanto IPv4, fornecidos através dos parâmetros.

acceptTCPConnection()

A função é responsável por estabelecer uma conexão TCP entre um servidor e um cliente. Para utilizá-la, é essencial primeiro criar um socket de servidor TCP, especificado como parâmetro. Uma vez que a conexão é estabelecida, este socket é utilizado para enviar e receber mensagens do cliente.

Estas três funções são essenciais para resolver o problema apresentado. Durante a implementação, foram incluídas diversas mensagens de erro e registros de atividades para auxiliar no diagnóstico de falhas que possam surgir durante a conexão. Na versão do código fornecida em anexo, a maioria dessas mensagens está comentada para atender aos padrões estabelecidos. No entanto, elas podem ser facilmente ativadas descomentando-as conforme necessário, proporcionando uma ferramenta adicional para monitoramento e análise.

2.1.1.3 Funções de manipulação de string

checkCommand()

A função analisa a validade de um comando ao comparar a primeira palavra do parâmetro fornecido com uma lista pré-definida de strings válidas. Essa verificação é crucial para determinar se o programa deverá ser encerrado ou não, baseando-se no comando recebido do input enviado pela unidade de monitoramento (cliente). Esta funcionalidade garante que apenas comandos apropriados sejam executados.

getNthCommand()

A função é projetada para extrair o enésimo comando de uma string específica, utilizando o espaço como delimitador para tokenizar a entrada. Ela opera com base em dois parâmetros: a string fornecida e a posição numérica (n) do comando a ser recuperado. Ao final do processo, a função retorna o enésimo comando identificado na string de entrada. Caso esse comando não esteja presente, a função retornará NULL, indicando a ausência do comando na posição especificada.

getInputDataFromString()

A função aceita dois parâmetros: uma string e um número inteiro 'n'. Utilizando espaços como delimitadores, ela tokeniza a string e a divide em duas partes: à esquerda e à direita do enésimo espaço especificado. Após completar essa divisão, a função retorna a porção da string localizada à direita do enésimo espaço. Este processo eficiente permite isolar segmentos específicos da string para análise ou manipulação subsequentes.

createClassroomStringData()

A função é projetada para receber dois parâmetros: um ponteiro para um vetor de salas e um indicador booleano que determina se a resposta deve incluir parênteses. Com base nesses parâmetros, ela manipula os dados contidos na estrutura Classroom, recuperando informações específicas sobre a sala e seus sensores. Ao final do processo, a função formata uma string de acordo com o padrão previamente definido nas especificações do sistema, garantindo que a saída esteja em conformidade com os requisitos necessários.

2.1.1.4 Funções específicas do projeto

executeCommand()

A função recebe uma string de comando e um ponteiro para a estrutura Classroom como parâmetro. Ela extrai o comando base da string de comando de entrada e executa diferentes ações com base neste comando base. Ela cria uma sala se o comando base for "CAD_REQ", ativa sensores com os valores fornecidos se o comando base for "INI_REQ", desliga os sensores se o comando base for "DES_REQ", atualiza os valores dos sensores se o comando base for "ALT_REQ", imprime os detalhes de uma sala especificada se o comando base for "SAL_REQ", imprime os detalhes de todas as salas se o comando base for "INF_REQ", e retorna NULL se o comando base não for reconhecido. A função retorna uma string como resultado de cada ação que será enviado para o cliente fazer a conversão em uma das respostas definidas na especificação do projeto.

translateResponseToMessage()

A função recebe um responseParameter como parâmetro e retorna uma mensagem correspondente com base no valor do parâmetro. Ela usa uma série de instruções if e else if para verificar se o responseParameter corresponde a strings específicas e retorna as mensagens correspondentes. Se nenhuma das condições for atendida, ela retorna o responseParameter como está, essa função está sendo utilizada no *client.c*.

checkInputAndCreatePayload()

A função aceita uma string como parâmetro de entrada e executa uma série de verificações e operações para criar uma string de carga útil baseada nessa entrada. Essencial no funcionamento do cliente, esta função é responsável por receber o input do usuário e verificar seu formato, uma etapa crucial para otimizar o processamento no servidor. Essa abordagem garante que apenas dados corretamente formatados sejam enviados, reduzindo erros e melhorando a eficiência da comunicação entre cliente e servidor.

2.1.1.5 Funções de checagem

Este grupo de funções é responsável por verificar e validar vários aspectos dos dados de sensores e salas em um sistema de monitoramento, elas fornecem mecanismos essenciais de validação e verificação de erros para os dados de sensores e salas no sistema, aqui está uma breve explicação do comportamento de cada função:

checkSensorsInputs()

Esta função verifica os dados de entrada dos sensores fornecidos como uma string. Primeiramente,

verifica se o número de elementos na string de entrada é igual a 6. Se não for, ela exibe uma mensagem de erro e retorna -1. Caso contrário, extrai e converte os dados de temperatura, umidade e ventilação da string de entrada. Em seguida, verifica se os valores estão nos intervalos válidos e se os dados do ventilador são válidos. Se todas as condições forem atendidas, ela retorna 1, indicando que os dados dos sensores são válidos. Caso contrário, exibe uma mensagem de erro e retorna -1.

checkRoomsSensorsAreInstalled()

Esta função verifica se os sensores estão instalados em uma sala específica. Ela recebe um array de objetos Classroom e um ID de sala como parâmetros. Primeiro, verifica se a sala existe no array. Se a sala existir, verifica se os estados de temperatura, umidade e ventilador naquela sala são todos -1, indicando que os sensores foram criados, mas não utilizados. Se for o caso, retorna 1. Se algum dos valores não for -1, retorna -2, indicando que os sensores estão sendo utilizados. Se a sala não existir, retorna -1. Para outros casos, retorna 0.

checkRoomCreated()

Esta função verifica se uma sala com um ID de sala fornecido já existe no array de salas de aula. Ela recebe o array de salas de aula e um ID de sala como parâmetros. Itera pelo array e compara o ID da sala com o ID de cada sala de aula. Se encontrar uma correspondência, retorna 1, indicando que a sala já existe. Se não encontrar uma correspondência, retorna -1, indicando que a sala não existe.

checkInputRegisterRoom()

Esta função verifica se o ID da sala de entrada é válido. Ela recebe o ID da sala como parâmetro de string. Converte o ID da sala em um inteiro e verifica se está dentro do intervalo válido de IDs de sala (de 0 a 7). Se o ID da sala for válido, retorna 1. Caso contrário, exibe uma mensagem de erro e retorna -1.

checkFanData()

Esta função verifica se os dados do ventilador são válidos. Ela recebe a posição do ventilador e um objeto Fan contendo o ID e o estado do ventilador como parâmetros. Verifica se o ID do ventilador corresponde à posição esperada e se o estado do ventilador está dentro do intervalo válido de 0 a 2. Se os dados do ventilador forem válidos, retorna 1. Caso contrário, retorna -1.

2.1.1.6 Funções de gerenciamento

Essas funções são relacionadas ao gerenciamento de salas de aula e seus valores de sensores. Aqui está uma breve explicação de cada função:

createRoomIfNotExists()

Esta função cria uma nova sala no array de salas de aula se ela ainda não existir. Ela verifica se a sala já existe. Se a sala existir, retorna uma mensagem de erro. Caso contrário, inicializa as propriedades da sala e retorna uma mensagem de sucesso.

updateSensorsValues()

Esta função atualiza os valores dos sensores para uma sala específica. Primeiro, verifica se a sala existe e se os sensores estão instalados. Se a sala existir e os sensores estiverem instalados, atualiza os valores dos sensores com base na string fornecida e retorna uma mensagem de sucesso. Se a sala não existir, os sensores não estiverem instalados ou ocorrer um erro desconhecido, retorna uma mensagem de erro.

turnOnSensorsValues()

Semelhante à updateSensorsValues(), esta função atualiza os valores dos sensores para uma sala específica. Também verifica se a sala existe e se os sensores estão instalados. Se a sala existir e os sensores estiverem instalados, atualiza os valores dos sensores e retorna uma mensagem de sucesso. Caso contrário, retorna uma mensagem de erro.

shutdownSensorsOfClassroom()

Esta função desliga os sensores de uma sala de aula. Verifica se a sala de aula especificada existe e se seus sensores estão instalados. Se a sala existir e os sensores estiverem instalados, atualiza os valores dos sensores para -1 (indicando desligamento) e retorna uma mensagem de sucesso. Se a sala existir mas os sensores não estiverem instalados, retorna uma mensagem de erro. Se a sala não existir ou ocorrer um erro desconhecido, retorna NULL.

2.1.1.7 Funções de exibição

printClassroom()

Esta função recebe um array de estruturas Classroom e um ID de sala como entrada. Ela verifica se a sala especificada existe e se seus sensores estão instalados. Se a sala existir e os sensores estiverem instalados, cria uma string de carga útil contendo os dados da sala e a retorna. A string de carga útil começa com "SAL_RES"e é seguida pelos dados de cada sala existente. Se a sala não existir, retorna a string "ERROR_03". Se os sensores não estiverem instalados, retorna a string "ERROR_06". Se ocorrer qualquer outro erro, retorna NULL.

printClassrooms()

Esta função recebe um array de estruturas Classroom como entrada. Ela verifica se alguma sala foi criada no array. Se não existirem salas, retorna a string "ERROR_03". Se existirem salas, cria uma string de carga útil contendo os dados de cada estrutura Classroom e a retorna. A string de carga útil começa com "INF_RES"e é seguida pelos dados de cada sala existente.

2.1.2 *client.c*

O módulo é a implementação principal de um programa cliente que utiliza sockets TCP para se comunicar com um servidor. O cliente envia comandos para o servidor e recebe respostas. O programa começa definindo várias variáveis, incluindo um buffer para dados de entrada. Em seguida, o programa verifica se o número correto de argumentos de linha de comando foi fornecido. Caso contrário, ele imprime uma mensagem de error e encerra. O primeiro argumento deve ser o nome do host (ou endereço IP) e o segundo argumento deve ser o número da porta. O programa então verifica se o nome do host fornecido é um endereço IP válido, caso não seja ele imprime um erro e encerra, então configura um socket de cliente TCP usando o nome do host e o número da porta fornecidos.

Após ele entra em um loop infinito para enviar e receber dados continuamente. Ele limpa o buffer e lê a entrada do usuário do input padrão. Se a entrada do usuário for um comando especial para sair do programa (conforme determinado pela função checkCommand), o programa encerra e fecha o socket, então sai do loop. Caso contrário, o programa verifica a entrada do usuário e cria uma carga útil usando a função checkInputAndCreatePayload, assim o programa envia a carga útil para o servidor. Se ocorrer um erro durante a criação do payload , ele chama a função de erro com uma mensagem de erro apropriada. O programa então limpa o buffer e recebe a resposta do servidor , por fim, o programa traduz a resposta para uma mensagem usando a função translateResponseToMessage e imprime a mensagem.

O programa continua em loop até que o usuário insira um comando especial para sair. No final do programa, ele retorna EXIT_SUCCESS para indicar uma execução bem-sucedida.

2.1.3 *server.c*

O módulo é a implementação principal de um programa servidor que utiliza sockets TCP para se comunicar com clientes. O servidor gerencia um conjunto de salas de aula, cada uma com seu próprio conjunto de sensores. O programa inicia definindo várias variáveis, incluindo um buffer para dados de entrada e um array de estruturas Classroom. A função initializeClassrooms() é então chamada para inicializar as salas de aula.

O programa verifica se os argumentos de linha de comando necessários são fornecidos. Se não, ele exibe uma mensagem de erro e encerra. O primeiro argumento deve ser "v4"ou "v6", indicando a versão de IP a ser usada. Se o primeiro argumento não for um desses, o programa exibe uma mensagem de erro e encerra

,em seguida, o programa configura um socket de servidor TCP usando o endereço IP e número de porta fornecidos (o segundo argumento de linha de comando). Ele então aceita uma nova conexão TCP.

O programa entra em um loop infinito para lidar com as solicitações dos clientes. Ele limpa o buffer, em seguida, recebe dados do cliente. Se ocorrer um erro durante a leitura, ele chama errorWithoutKill() com uma mensagem de erro apropriada. Se o cliente encerrar a conexão, o programa sai do loop. Se o comando recebido for "kill", o programa exibe uma mensagem e sai do loop. Caso contrário, ele exibe a solicitação recebida, executa o comando usando a função executeCommand() e envia o resultado de volta para o cliente. Se ocorrer um erro durante a escrita, ele chama errorWithoutKill() com uma mensagem de erro apropriada.

Finalmente, o programa fecha os sockets e retorna EXIT_SUCCESS para indicar uma execução bem-sucedida.

3 Discussão e conclusão

O uso de redes para interconectar programas é um aspecto fascinante do desenvolvimento de software. Abordar esse conceito em uma disciplina acadêmica proporciona uma compreensão mais clara e aplicada do tema. Como se trata do primeiro trabalho da disciplina, acredito que foi escolhido não aprofundar demasiadamente nos detalhes técnicos do desenvolvimento de redes, focando mais em uma visão geral sobre o comportamento das redes e como elas podem ser aplicadas em sistemas reais. Esta abordagem inicial nos permitiu explorar as bases fundamentais das redes de computadores, preparando o terreno para investigações mais detalhadas em trabalhos futuros.

A experiência de nos aprofundar pouco nas redes de computadores durante esse trabalho prático teve um sabor agridoce. Por ser uma disciplina introdutória e este ser apenas o primeiro trabalho, entendo que não entramos com muitos detalhes sobre como a biblioteca de sockets da linguagem funciona. Tivemos uma visão geral usando buffer de um tamanho definido e utilizando os métodos ‘send‘ e ‘recv‘ da biblioteca, o que despertou o interesse em explorar outras funcionalidades e protocolos. No entanto, percebi que a maior parte do tempo foi dedicada à manipulação de strings, um tópico já abordado em disciplinas anteriores e que não impacta diretamente no entendimento das redes. Esse foi o aspecto que mais exigiu tempo, embora não tenha sido o mais desafiador.

Ao concluir este trabalho, meu interesse por redes de computadores aumentou significativamente, junto com ideias de como aprimorar a solução desenvolvida. Estou ansioso pelo próximo projeto da disciplina, esperando que ele aproveite o que foi desenvolvido aqui e talvez introduza requisitos mais complexos, como aceitar múltiplas conexões, organizar salas por edifícios ou estabelecer conexões P2P entre clientes. Essas são algumas das possibilidades que estou entusiasmado para explorar.