Universidade de Brasília Faculdade UnB Gama

Projeto de Pesquisa

Fundamentos de Redes de Computadores
Prof. : Fernando William Cruz
Engenharia de Software

Alunos
Ana Carolina Rodrigues Leite - 19/0101792
Ricardo de Castro Loureiro - 20/0043111

1. Introdução

No contexto das redes de computadores e da arquitetura TCP/IP, a compreensão da arquitetura de aplicações que envolvem a gerência de diálogo é fundamental. Nesse sentido, este projeto tem como objetivo desenvolver uma aplicação que permite aos alunos explorarem e compreenderem essa arquitetura por meio da criação de salas de bate-papo virtuais.

A comunicação em rede é essencial para a troca de informações e o compartilhamento de recursos entre dispositivos conectados. A arquitetura TCP/IP é amplamente utilizada como base para a comunicação na internet e em redes locais, fornecendo uma estrutura padronizada para a transferência de dados.

A gerência de diálogo é uma parte crucial da arquitetura de aplicações de rede. Envolve a capacidade de estabelecer, manter e finalizar conexões entre os participantes da comunicação. Em uma aplicação de bate-papo, por exemplo, é necessário permitir que os clientes ingressem em salas virtuais e interajam entre si de forma eficiente e confiável.

A solução proposta neste relatório é a construção de uma aplicação de salas de bate-papo virtuais. Os alunos terão a oportunidade de explorar os aspectos da arquitetura TCP/IP relacionados à gerência de diálogo, como o estabelecimento de conexões, o controle de fluxo e o tratamento de erros. Além disso, poderão compreender conceitos como sockets, protocolos de transporte (como TCP e UDP) e protocolos de aplicação (como HTTP).

Através dessa aplicação, os alunos poderão aprofundar seus conhecimentos teóricos, colocando-os em prática na implementação de uma solução real. A interação em salas de bate-papo virtuais proporcionará uma experiência prática de comunicação em rede, permitindo aos alunos compreenderem os desafios e as nuances envolvidas nesse tipo de aplicação.

No decorrer deste relatório, serão abordados os principais conceitos e componentes da arquitetura de aplicações de rede, juntamente com a descrição detalhada da implementação da aplicação de salas de bate-papo virtuais. Espera-se que essa experiência contribua para o aprimoramento do conhecimento dos alunos sobre a arquitetura TCP/IP e suas aplicações.

2. Metodologia utilizada

Para realizar o desenvolvimento da aplicação de salas de bate-papo virtuais e a elaboração do relatório final, a dupla adotou a seguinte metodologia:

Revisão dos códigos disponibilizados pelo professor: Dedicamos um tempo inicial para revisar os códigos fornecidos pelo professor, a fim de compreender melhor o funcionamento das funções necessárias para a implementação do programa de chat. Essa revisão permitiu identificar os componentes-chave e entender as etapas necessárias para o desenvolvimento do projeto.

Seções de programação em pares: Realizamos duas sessões de programação em pares, cada uma com aproximadamente três horas de duração. Durante essas sessões, os membros da equipe colaboraram ativamente, compartilhando conhecimentos e experiências para implementar a aplicação de salas de bate-papo virtuais. Essa abordagem de programação em pares ajudou a promover o aprendizado conjunto e a resolver desafios de forma mais eficiente.

Reunião feita pelo discord: A fim de elaborar o relatório final e preparar a apresentação do projeto, a dupla se reuniu em duas chamadas pelo discord. Nessas reuniões, discutimos a estrutura e o conteúdo do relatório, revisamos os resultados obtidos, compartilhamos ideias e revisamos o trabalho realizado. A comunicação pelo discord permitiu uma interação mais direta e eficiente, facilitando a troca de informações e o planejamento das atividades.

A combinação dessas etapas permitiu à dupla desenvolver a aplicação de salas de bate-papo virtuais de acordo com os princípios da arquitetura TCP/IP, bem como produzir um relatório abrangente e coerente, documentando o processo de desenvolvimento e os resultados alcançados.

3. Descrição da solução

O código implementa um servidor de bate-papo em C usando sockets e a chamada select(). A solução usa duas structs e oito funções para organizar o código. Os clientes podem se conectar, escolher salas existentes ou criar novas salas, enviar mensagens para os participantes da sala e executar outras operações básicas. Cada função desempenha um papel específico, como tratar novas conexões, processar mensagens dos clientes e gerenciar as salas. O resumo destaca a estrutura e as principais funcionalidades do servidor de bate-papo implementado no código.

```c
typedef struct {
    int id;
    char name[MAX_NAME_LENGTH];
    int socket;
} Client;

typedef struct {
    int id;
    char name[MAX_NAME_LENGTH];
    int limit;
    int num_clients;
    Client* clients[MAX_CLIENTS];
} Room;

Room* rooms[MAX_ROOMS];
int num_rooms = 0;
int max_sd; // Valor máximo do descritor de arquivo (socket) para a chamada select()
fd_set read_fds; // Conjunto de descritores de arquivo para leitura
```

Nessa parte do código, são definidas duas structs: Client e Room. A struct Client armazena informações sobre cada cliente conectado, como o ID, nome e o socket. A struct Room representa as salas de bate-papo, contendo campos para o ID, nome, limite de clientes, número atual de clientes e um vetor de ponteiros para os clientes presentes na sala.

Além disso, são declaradas variáveis globais, como rooms (vetor de ponteiros para Room), num_rooms (número atual de salas), max_sd (valor máximo do descritor de arquivo para a chamada select()) e read_fds (conjunto de descritores de arquivo para leitura).

Essa estrutura permite gerenciar clientes e salas no servidor de bate-papo, organizando as informações em structs e usando variáveis globais para controle. As funções subsequentes utilizarão essa estrutura para controlar as conexões e interações entre os clientes e as salas.

```c
void send_message(int socket, const char* message) {
    write(socket, message, strlen(message));
}
```

A função send_message envia uma mensagem para um socket de cliente específico. Ela recebe o socket de destino e a mensagem como parâmetros. Utilizando a função write, os dados da mensagem são escritos no socket. Essa função é útil para enviar mensagens de texto do servidor para um cliente em particular, utilizando o socket como meio de comunicação. Ao chamar essa função, a mensagem é enviada ao cliente correspondente, possibilitando a troca de informações entre o servidor e os clientes conectados.

```c
void send_clients_list(int socket, Room* room) {
    char clients_list[1000];
    sprintf(clients_list, "===== Clientes Conectados Na Sala =====\n");
    for (int i = 0; i < room->num_clients; i++) {
        sprintf(clients_list + strlen(clients_list), "%s\n", room->clients[i]->name);
    }
    send_message(socket, clients_list);
}
```

A função send_clients_list tem como objetivo enviar para um socket de cliente a lista de clientes conectados a uma determinada sala. Ela recebe o socket de destino e um ponteiro para a estrutura da sala como parâmetros. A função formata e constrói a lista de clientes, adicionando o nome de cada cliente à medida que percorre a lista de clientes da sala. Em seguida, utiliza a função send_message para enviar a lista para o socket de destino. Essa função é útil para fornecer ao cliente uma visão atualizada dos clientes presentes na sala, permitindo interações e visualizações dentro do ambiente compartilhado.

```c
Room* find_room_by_id(int room_id) {
    for (int i = 0; i < num_rooms; i++) {
        if (rooms[i]->id == room_id) {
            return rooms[i];
        }
    }
    return NULL;
}
```

A função find_room_by_id busca uma sala específica com base em seu ID. Ela percorre um array de salas e compara o ID de cada sala com o ID fornecido como argumento. Se houver correspondência, a função retorna um ponteiro para a sala encontrada. Caso contrário, retorna NULL. Essa função é útil para localizar salas

específicas e realizar operações relacionadas a elas.

```c
void strtrim(char* str) {
    int start = 0, end = strlen(str) - 1;

    while (isspace((unsigned char)str[start])) {
        start++;
    }

    if (start == strlen(str)) {
        str[0] = '\0';
        return;
    }

    while (isspace((unsigned char)str[end])) {
        end--;
    }

    int i, j;
    for (i = start, j = 0; i <= end; i++, j++) {
        str[j] = str[i];
    }
    str[j] = '\0';
}
```

A função strtrim remove os espaços em branco desnecessários de uma string, tanto no início quanto no final. Ela percorre a string em busca do primeiro e último caractere não vazio, eliminando os espaços em branco entre eles. Em seguida, copia os caracteres válidos para uma nova posição na string. Essa função é útil para limpar e processar strings, facilitando sua manipulação e comparação.

```c
Room* create_room(int room_id, const char* room_name, int limit) {
    if (num_rooms >= MAX_ROOMS) {
        return NULL;
    }

    Room* new_room = (Room*)malloc(sizeof(Room));
    new_room->id = room_id;
    strcpy(new_room->name, room_name);
    new_room->limit = limit;
    new_room->num_clients = 0;
    rooms[num_rooms] = new_room;
    num_rooms++;

    return new_room;
}
```

A função create_room cria uma nova sala e a adiciona à lista de salas existentes. Ela recebe um identificador, um nome e um limite de clientes como parâmetros. Primeiro, verifica se o limite máximo de salas foi atingido. Em seguida, aloca a memória para a nova sala e preenche os campos com os valores fornecidos. A sala é adicionada à lista e o contador de salas é incrementado. Por fim, o ponteiro para a nova sala é retornado. Essa função permite criar e gerenciar dinamicamente salas em um sistema de compartilhamento de ambientes.

```
void remove_client_from_room(Client* client, Room* room) {
    for (int i = 0; i < room->num_clients; i++) {
        if (room->clients[i] == client) {
            for (int j = i; j < room->num_clients - 1; j++) {
                room->clients[j] = room->clients[j + 1];
            }
            room->num_clients--;
            break;
        }
    }
}
```

A função remove_client_from_room remove um cliente específico de uma sala. Ela percorre a lista de clientes da sala e, ao encontrar o cliente desejado, reorganiza a lista para preencher o espaço vago. Em seguida, o número de clientes na sala é atualizado. Essa função é útil para gerenciar a lista de clientes de uma sala, permitindo a remoção de um cliente em casos como desconexão ou saída da sala.

```
void handle_client_message(Client* client, const char* message) {
    Room* current_room = NULL;
    for (int i = 0; i < num_rooms; i++) {
        Room* room = rooms[i];
        for (int j = 0; j < room->num_clients; j++) {
            if (room->clients[j] == client) {
                current_room = room;
                break;
            }
        }
    }

    if (strncmp(message, "/listar", 7) == 0) {
        send_clients_list(client->socket, current_room);
    } else if (strncmp(message, "/sair", 5) == 0) {
        if (current_room != NULL) {
            remove_client_from_room(client, current_room);
            send_message(client->socket, "Você saiu da sala.\n");
        } else {
            send_message(client->socket, "Você não está em nenhuma sala.\n");
        }
    } else if (strncmp(message, "/trocar_sala", 12) == 0) {
        if (current_room != NULL) {
            remove_client_from_room(client, current_room);
            current_room = NULL;
        }
```

```
    int room_id = atoi(message + 13);
    if (room_id >= 0) {
        Room* new_room = find_room_by_id(room_id);
        if (new_room != NULL) {
            if (new_room->num_clients < new_room->limit) {
                new_room->clients[new_room->num_clients++] = client;
                current_room = new_room;
                send_message(client->socket, "Você entrou na nova sala.\n");
            } else {
                send_message(client->socket, "A sala está cheia.\n");
            }
        } else {
            send_message(client->socket, "Sala não encontrada.\n");
        }
    } else if (room_id == -1) {
        send_message(client->socket, "ID da sala não pode ser -1. Digite outro ID válido.\n");
    } else {
        send_message(client->socket, "ID da sala inválido. Digite um ID válido.\n");
    }
} else {
    if (current_room != NULL) {
        for (int i = 0; i < current_room->num_clients; i++) {
            char formatted_message[1100];
            sprintf(formatted_message, "[%s] => %s", client->name, message);
```

```
                // Envie a mensagem formatada em uma única linha para todos os clientes na sala
                send_message(current_room->clients[i]->socket, formatted_message);
            }
        } else {
            send_message(client->socket, "Você não está em nenhuma sala.\n");
        }
    }
}
```

A função handle_client_message é responsável por processar as mensagens enviadas por um cliente. Ela verifica o conteúdo da mensagem e executa ações correspondentes, como enviar a lista de clientes conectados, remover o cliente de uma sala, trocar de sala ou enviar mensagens para outros clientes na sala. Além disso, a função mantém o controle da sala atual do cliente. Essa função é essencial para o funcionamento do sistema de chat, permitindo que os clientes interajam e se comuniquem de acordo com as regras e comandos estabelecidos.

```
void handle_new_connection(int server_socket) {
    int client_socket;
    struct sockaddr_in client_address;
    socklen_t client_address_len = sizeof(client_address);

    client_socket = accept(server_socket, (struct sockaddr*)&client_address, &client_address_len);
    if (client_socket == -1) {
        perror("Erro ao aceitar a conexão");
        return;
    }

    printf("Cliente conectado\n");

    Client* client = (Client*)malloc(sizeof(Client));
    client->id = client_socket;
    memset(client->name, 0, MAX_NAME_LENGTH);
    client->socket = client_socket;

    send_message(client->socket, "Insira seu nome:\n");
    read(client->socket, client->name, MAX_NAME_LENGTH);

    // Remova o espaço em branco no final do nome
    strtrim(client->name);

    send_message(client->socket, "Insira o ID da sala (-1 para criar uma nova):\n");
```

FIGURA 5: Tratamento da conexão de um novo cliente ao servidor.

Fonte: Autor.

A função handle_new_connection desempenha um papel fundamental no tratamento da conexão de um novo cliente ao servidor. Quando um cliente se conecta, a função é chamada para lidar com os procedimentos iniciais.

Primeiramente, a função utiliza a chamada de sistema accept para aceitar a conexão e obter o descritor de arquivo do socket do cliente. Em seguida, solicita ao cliente que insira seu nome por meio da função send_message e lê a resposta usando a função read. O nome é armazenado na estrutura Client correspondente ao cliente recém-conectado.

Em seguida, a função pede ao cliente que insira o ID da sala desejada. Com base nesse ID, a função busca a sala correspondente usando a função find_room_by_id. Se a sala existir e tiver espaço disponível para o cliente, o cliente é

adicionado à sala através do incremento de num_clients e a mensagem de entrada na sala é enviada aos outros clientes presentes.

Caso o ID da sala seja -1, indicando a criação de uma nova sala, a função solicita ao cliente que insira o limite de participantes da sala. Com base nesse limite, a função chama a função create_room para criar uma nova sala com um ID único e adiciona o cliente à sala recém-criada.

A função envia mensagens de confirmação ou erro ao cliente, informando se ele entrou em uma sala existente, na nova sala criada ou se houve algum problema, como a sala está cheia ou não foi encontrada.

Por fim, a função atualiza o conjunto de descritores de arquivo para leitura, adicionando o descritor de arquivo do novo cliente. Também atualiza o valor máximo do descritor de arquivo, caso o número do cliente seja maior.

Em resumo, a função handle_new_connection gerencia a autenticação e atribuição de clientes a salas, garantindo a comunicação adequada e o controle do servidor. Ela desempenha um papel fundamental na interação inicial entre o servidor e os clientes, permitindo que os clientes entrem em salas existentes ou criem novas salas, conforme a necessidade.

```c
int main(int argc, char* argv[]) {
    if (argc != 3) {
        printf("Uso: %s [IP] [PORTA]\n", argv[0]);
        return 1;
    }

    int server_socket, client_socket, max_clients = 0;
    int activity, i, valread, sd;
    struct sockaddr_in server_address;
    socklen_t client_address_len = sizeof(server_address);
    char buffer[1000];

    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Erro ao criar o socket");
        return 1;
    }

    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr(argv[1]);
    server_address.sin_port = htons(atoi(argv[2]));

    if (bind(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == -1) {
        perror("Erro ao associar o socket ao endereço");
        return 1;
    }
```

A função main inicia o servidor, verificando os argumentos de linha de comando para obter o endereço IP e a porta para vincular o socket. Em seguida, cria o socket,

associa-o ao endereço e o coloca no modo de escuta por conexões.

Em um loop infinito, o servidor aguarda atividade nos descritores de arquivo usando a função select. Se houver atividade no socket do servidor, indica que um novo cliente se conectou e chama a função handle_new_connection para tratar a conexão.

Em seguida, o servidor verifica se há atividade nos descritores de arquivo dos clientes conectados às salas existentes. Se houver atividade, lê a mensagem enviada pelo cliente e chama a função handle_client_message para processar a mensagem.

Se ocorrer uma desconexão de um cliente, o servidor fecha o descritor do arquivo, remove o cliente da sala correspondente e atualiza o conjunto de descritores de arquivo para leitura.

Em resumo, o código implementa um servidor de bate-papo que permite a conexão de vários clientes e a comunicação entre eles por meio de salas de bate-papo. Ele utiliza a função select para monitorar a atividade nos sockets e executa as operações apropriadas para tratar as conexões e as mensagens dos cliente.

4. Conclusão

4.1 - Resultado do projeto

O projeto implementa um servidor de bate-papo em C, permitindo a comunicação entre vários clientes. Ele demonstra a aplicação prática de sockets e a função select para lidar com conexões simultâneas. O servidor suporta a criação de múltiplas salas de bate-papo, com limite de participantes configurável. Os clientes podem entrar em salas existentes, criar novas salas e trocar de sala durante a sessão. O código gerencia adequadamente a adição e remoção de clientes nas salas, garantindo que as capacidades sejam respeitadas. Além disso, as mensagens enviadas pelos clientes são transmitidas para todos os participantes da sala atual. O projeto apresenta uma estrutura organizada e modular, separando as funcionalidades em funções distintas para facilitar a manutenção e o entendimento do código. Em resumo, o servidor de bate-papo implementado é uma solução robusta e funcional para comunicação em tempo real entre múltiplos clientes.

4.2 - Considerações dos membros

Ana Carolina Rodrigues: O trabalho me proporcionou um aprendizado muito grande sobre como funciona a conexão TCP/IP de maneira mais interativa e no funcionamento de servidores para poder gerenciar a troca de informações entre os clientes. E ademais sobre o fato dos diversos comandos que se pode optar em colocar no código, configurado a api do select. Entendo que a nota 9.0 seria condizente com minha participação.

Ricardo de Castro Loureiro: A experiência deste trabalho proporcionou uma aprendizagem extremamente interessante sobre o funcionamento dos servidores utilizados para facilitar a troca de informações. Além disso, tive a oportunidade de adquirir conhecimento sobre o uso do comando *select()*, que possibilita que um sistema monitore múltiplos descritores de arquivo. O aspecto mais satisfatório desse trabalho foi a possibilidade de aplicar os conceitos teóricos aprendidos em sala de aula e lidar com os desafios práticos que surgiram. Uma nota justa seria 9.5 pelo trabalho desenvolvido.

5.0 Referência Bibliografia

[1] CRUZ, F. W. System Call Select(). Disponível em: <https://aprender3.unb.br/pluginfile.php/2561365/mod_resource/content/1/selectserver.c>. Acesso em: 29 Julho de 2023.

[2] PRACTICALS, C. S. E. Learn Select system call in CCSE Practicals, Disponível em: <https://www.youtube.com/watch?v=CfEShMmsUus>. Acesso em: 29 Julho de 2023.

[3] Man7. select(2) — Linux manual page. Disponível em: <https://man7.org/linux/man-pages/man2/select.2.html>. Acesso em: 28 Julho de 2023.

5.0 Links

Link do repositório com código fonte: https://github.com/castroricardo1/Projeto-Redes