UNIVERSIDADE FEDERAL DE MINAS GERAIS

REDES DE COMPUTADORES

Trabalho Prático 02

JOÃO VITOR FERREIRA / 2021039654

BELO HORIZONTE

2024

SUMÁRIO

CODIGO SERVIDOR

Este servidor, implementado em C e utilizando o protocolo UDP (User Datagram Protocol), foi projetado para simular a funcionalidade básica de aplicativos de streaming. Ele é capaz de criar um socket e escutar por conexões de múltiplos clientes simultaneamente. Quando uma conexão for estabelecida, o servidor recebe a escolha de filme feita pelo cliente e inicia o envio das citações desse filme para o cliente, enviando uma citação a cada 3 segundos. Além disso, a cada 4 segundos, o servidor imprime a quantidade atual de clientes conectados. Caso haja alguma recusa por parte do cliente, o servidor notifica ao cliente sobre a recusa e encerra a respectiva conexão. Este servidor é compatível tanto com o protocolo IPv4 quanto com o IPv6.

Funcionamento do Servidor

`handle_client_error(int bytes_received)`: Esta função verifica se houve algum erro durante a recepção de dados de um cliente usando a função `recvfrom()`. Caso um erro seja detectado (`bytes_received == -1`), uma mensagem de erro é exibida e o thread atual é encerrado usando `pthread_exit()`.[figura 1]

```c
// Function to handle errors when receiving data from a client.
// Prints an error message and exits the current thread.
void handle_client_error(int bytes_received)
{
    if (bytes_received == -1)
    {
        perror("Erro ao receber dados do cliente");
        pthread_exit(NULL); // Exit the thread if there's an error
    }
}
```

Figura 1

`increment_client_count()`: o contador global de clientes (client_count) de forma segura para threads. A função usa um mutex (lock) para garantir que apenas um thread modifique o contador por vez. [figura 2]

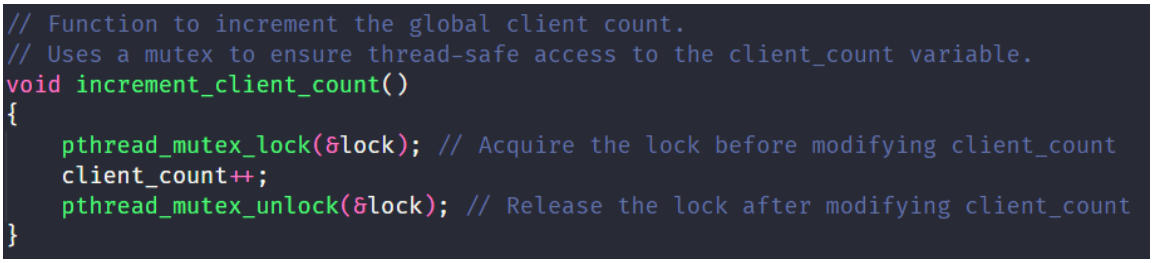


Figura 2

`decrement_client_count()`: Extremamente similar a função `increment_client_count()` Decrementa o contador global de clientes (client_count) de forma segura para threads. A função usa um mutex (lock) para garantir que apenas um thread modifique o contador por vez. [figura 3]

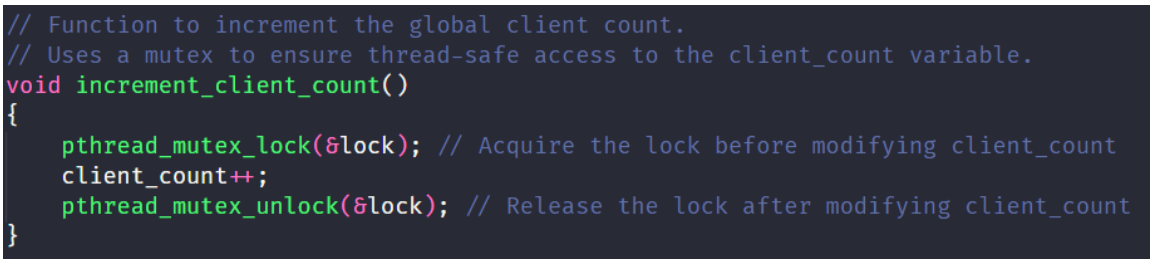


Figura 3

`handle_client(void *arg)`: Esta função lida com a conexão de um cliente. As etapas executadas são:

- **Extrai as informações do cliente:** A função recebe um ponteiro para uma estrutura `client_info` como argumento. Essa estrutura contém o descritor do socket do servidor (`socket`), o filme escolhido pelo cliente (`chosen_movie`) e o índice da citação atual (`quote_from_the_movie`). [figura 4]

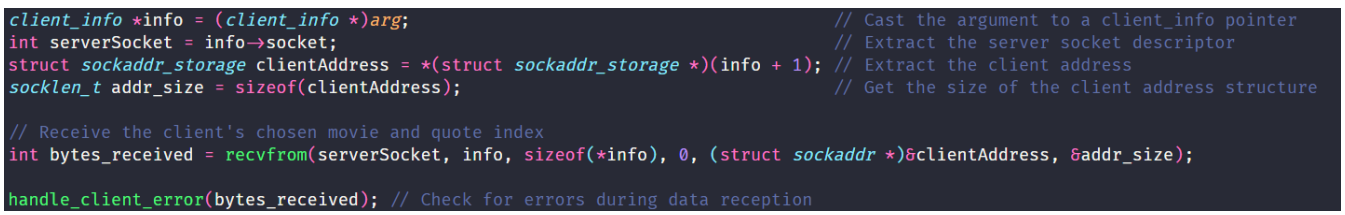


Figura 4

- **Incrementa o contador de clientes:** Usa a função `increment_client_count()` para registrar o novo cliente. [figura 5]

```
increment_client_count(); // Increment the count of active clients
```

Figura 5

- **Envia as citações:** Itera pelas citações do filme escolhido e as envia para o cliente usando a função `sendto()`. Um intervalo de 3 segundos é introduzido entre o envio de cada citação usando `sleep(3)`. [figura 6]

```
// Iterate through and send quotes from the chosen movie
for (int i = 0; i < NUM_QUOTES; i++)
{
    char *sentence = movie_sentences[info->chosen_movie - 1][i];                                    // Get the quote based on the chosen movie index
    sendto(serverSocket, sentence, strlen(sentence) + 1, 0, (struct sockaddr *)&clientAddress, addr_size); // Send the quote to the client
    sleep(3);                                                                                       // Introduce a 3-second delay between sending quotes
}
```

Figura 6

- **Decrementa o contador de clientes:** Usa a função `decrement_client_count()` para registrar a desconexão do cliente. [figura 7]

```
decrement_client_count(); // Decrement the count of active clients
```

Figura 7

- **Libera a memória:** Libera a memória alocada para a estrutura `client_info` usando `free()`.[figura 8]

```
free(info); // Release the memory allocated for the client_info structure
```

Figura 8

- **Encerra o thread:** Finaliza o thread atual. [figura 9]

```c
return NULL; // Exit the thread
```

Figura 9

`create_server_socket_and_listen(sa_family_t family, int port)`: Esta função cria um socket UDP e o associa a uma porta específica. Ela também entra em um loop para ouvir conexões de clientes e cria uma thread para lidar com cada conexão. As etapas executadas são:

- **Cria um socket UDP:** A função `socket()` é usada para criar um socket UDP usando o protocolo SOCK_DGRAM. [figura 10]

```c
serverSocket = socket(family, SOCK_DGRAM, 0); // Create a UDP socket
```

Figura 10

- **Configura o endereço do servidor para aceitar conexões IPv4 ou IPv6:** É definida a família de endereços, o número da porta e o endereço IP na estrutura de endereço apropriada (sockaddr_in para IPv4 e sockaddr_in6 para IPv6). O endereço IP é definido para aceitar qualquer endereço disponível. [figura 11]

```c
// Configure the server address based on the address family
if (family == AF_INET)
{
    ((struct sockaddr_in *)&serverAddress)->sin_family = family;          // Set address family to IPv4
    ((struct sockaddr_in *)&serverAddress)->sin_port = htons(port);       // Set the port number (converted to network byte order)
    ((struct sockaddr_in *)&serverAddress)->sin_addr.s_addr = INADDR_ANY; // Bind to any available IPv4 address
    addr_size = sizeof(struct sockaddr_in);
}
else if (family == AF_INET6)
{
    ((struct sockaddr_in6 *)&serverAddress)->sin6_family = family;                // Set address family to IPv6
    ((struct sockaddr_in6 *)&serverAddress)->sin6_port = htons(port);             // Set the port number (converted to network byte order)
    inet_pton(family, "::", &(((struct sockaddr_in6 *)&serverAddress)->sin6_addr)); // Bind to any available IPv6 address
    addr_size = sizeof(struct sockaddr_in6);
}
```

Figura 11

- **Associa o socket a uma porta:** A função `bind()` é usada para associar o socket à porta e família de endereços especificadas nos argumentos. [figura 12]

```c
// Bind the socket to the address and port
if (bind(serverSocket, (struct sockaddr *)&serverAddress, addr_size) == -1)
{
    perror("Erro ao associar o socket ao endereço");
    exit(EXIT_FAILURE);
}
```

Figura 12

- **Entra em um loop de escuta:** Um loop infinito é usado para ouvir continuamente as conexões de clientes. [figura 13]

```c
// Main loop to listen for incoming connections
while (1)
{
    struct sockaddr_storage clientAddress;
    socklen_t clientAddressLength = sizeof(clientAddress);

    // Wait for data to arrive from a client (blocking call)
    recvfrom(serverSocket, NULL, 0, MSG_PEEK, (struct sockaddr *)&clientAddress, &clientAddressLength);

    // Create a new thread to handle the client connection
    pthread_t tid;
    client_info *info = malloc(sizeof(client_info) + sizeof(struct sockaddr_storage)); // Allocate memory for client_info and clientAddress
    info->socket = serverSocket;                                                       // Store the server socket descriptor in client_info

    memcpy(info + 1, &clientAddress, sizeof(clientAddress)); // Copy the client address into the allocated memory
    pthread_create(&tid, NULL, handle_client, info);         // Create a new thread and pass the client_info as an argument
}
```

Figura 13

- **Espera por dados:** A função `recvfrom()` é usada em modo de espiada (`MSG_PEEK`) para verificar se há dados de um cliente sem realmente removê-los da fila. [figura 14]

```c
// Wait for data to arrive from a client (blocking call)
recvfrom(serverSocket, NULL, 0, MSG_PEEK, (struct sockaddr *)&clientAddress, &clientAddressLength);
```

Figura 14

- **Cria um thread para o cliente:** Se dados forem detectados, um novo thread é criado usando `pthread_create()`. A função `handle_client()` é passada como a função do thread, e uma estrutura `client_info` contendo as informações do cliente é passada como argumento. [figura 15]

```c
// Create a new thread to handle the client connection
pthread_t tid;
client_info *info = malloc(sizeof(client_info) + sizeof(struct sockaddr_storage)); // Allocate memory for client_info and clientAddress
info->socket = serverSocket;                                                       // Store the server socket descriptor in client_info
```

Figura 15

`print_client_count(void *arg)`: Esta função imprime o número atual de clientes conectados a cada 4 segundos. Ela usa um `mutex (lock)` para garantir acesso seguro à variável `client_count`. [figura 16]

```c
void *print_client_count(void *arg)
{
    while (1)
    {
        sleep(4); // Sleep for 4 seconds
        pthread_mutex_lock(&lock);              // Acquire the lock before accessing client_count
        printf("Clientes: %d\n", client_count); // Print the client count
        pthread_mutex_unlock(&lock);            // Release the lock after accessing client_count
    }
}
```

Figura 16

`main(int argc, char **argv)`: A função `main()` é o ponto de entrada do programa. As etapas executadas são:

- **Valida os argumentos da linha de comando:** Verifica se o número de argumentos está correto e extrai a versão do `IP (IPv4 ou IPv6)` e o número da porta. [figura 17]

```c
if (argc ≠ 3)
{
    printf("Número inválido de argumentos\n");
    exit(EXIT_FAILURE);
}

char *ipVersion = argv[1]; // Get the IP version (e.g., "ipv4" or "ipv6")
int port = atoi(argv[2]);  // Convert the port number to an integer

// Validate the IP version
if (strcmp(ipVersion, "ipv4") ≠ 0 && strcmp(ipVersion, "ipv6") ≠ 0)
{
    printf("Versão de IP inválida\n");
    exit(EXIT_FAILURE);
}
```

Figura 17

- **Inicializa o mutex:** A função `pthread_mutex_init()` é usada para inicializar o `mutex lock`. [figura 18]

```c
pthread_mutex_init(&lock, NULL); // Initialize the mutex
```

Figura 18

- **Cria um thread para imprimir o contador de clientes:** Esta parte do código cria um thread usando `pthread_create()`. A função `print_client_count()` é passada como a função do thread, o que significa que este thread executará o código dentro de `print_client_count()`. O argumento `NULL` indica que os atributos de thread padrão serão usados. [figura 19]

```
pthread_t tid;
pthread_create(&tid, NULL, print_client_count, NULL); // Create a thread to print the client count
```

Figura 19

- **Inicia o servidor:** Este bloco de código condicional inicia o servidor com base na versão do IP especificada nos argumentos da linha de comando. Se a versão for `"ipv4"`, a função `create_server_socket_and_listen()` é chamada com a família de endereços AF_INET e o número da porta. Se a versão for `"ipv6"`, a mesma função é chamada com `AF_INET6`. [figura 20]

```
// Start the server based on the IP version
if (strcmp(ipVersion, "ipv4") == 0)
{
    create_server_socket_and_listen(AF_INET, port); // Start the server in IPv4 mode
}
else if (strcmp(ipVersion, "ipv6") == 0)
{
    create_server_socket_and_listen(AF_INET6, port); // Start the server in IPv6 mode
}
```

Figura 20

Detalhes da Conexão

- `socket()`[figura 10]: Esta função é utilizada para criar um socket. Ela recebe três argumentos: a família de endereços (por exemplo, AF_INET para IPv4), o tipo de socket (por exemplo, SOCK_DGRAM para UDP) e o protocolo (geralmente 0 para o protocolo padrão). A função retorna um descritor de socket que pode ser usado em chamadas de função subsequentes que operam em sockets.
- `bind()`[figura 12]: Esta função associa o socket criado a um endereço específico e porta. O primeiro argumento é o descritor do socket, o segundo

argumento é um ponteiro para a estrutura de endereço que contém detalhes do endereço e porta, e o terceiro argumento é o tamanho da estrutura de endereço. Isso permite que o servidor receba conexões de clientes nesse endereço e porta especificados.

- `sendto()`[figura 6]: Esta função é usada para enviar dados através do socket para o cliente conectado. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para os dados a serem enviados, o terceiro argumento é o tamanho dos dados, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes enviados ou -1 em caso de erro.
- `recvfrom()`[figura 13 e 14]:Esta função é usada para receber dados do cliente conectado através do socket. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para um buffer onde os dados serão armazenados, o terceiro argumento é o tamanho do buffer, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes recebidos ou -1 em caso de erro.
- `close()`[figura 15]: Esta função é usada para fechar um descritor de socket, liberando os recursos associados a ele.

CODIGO CLIENTE

Este cliente implementa um serviço simples para se conectar a um servidor de filmes e receber citações. Ele utiliza o protocolo UDP (User Datagram Protocol) para comunicação com o servidor. O cliente permite que o usuário escolha um filme de uma lista e, em seguida, recebe e exibe as citações desse filme, uma por vez, com um intervalo de tempo entre cada recebimento. Este cliente suporta tanto o protocolo IPv4 quanto o IPv6.

Funcionamento do Servidor

`create_socket(sa_family_t family)`: Esta função cria um socket UDP. O argumento `family` especifica a família de endereços (`IPv4 ou IPv6`). A função usa a chamada `socket()` para criar o socket e retorna o descritor do socket. Em caso de erro, uma mensagem de erro é exibida e o programa é encerrado. [figura 21]

```c
int create_socket(sa_family_t family)
{
    int clientSocket = socket(family, SOCK_DGRAM, 0); // Create a UDP socket
    if (clientSocket == -1)
    {
        perror("Erro ao criar o socket"); // Print an error message if socket creation fails
        exit(EXIT_FAILURE);               // Exit the program if there's an error
    }
    return clientSocket; // Return the socket descriptor on success
}
```

Figura 21

`connect_socket(int clientSocket, sa_family_t family, char *ipAddress, int port, client_info info)`: Esta função conecta o socket do cliente ao servidor e envia as informações do cliente. As etapas executadas são:

- **Configura a estrutura de endereço do servidor:** A função `memset()` é usada para limpar a estrutura `serverAddress`. Em seguida, a estrutura é configurada com base na família de endereços (`family`). Para `IPv4`, a estrutura `sockaddr_in` é usada, enquanto para IPv6, a estrutura `sockaddr_in6` é usada. A função `inet_pton()` converte a string do endereço `IP` para a forma binária e a armazena na estrutura. [figura 22]

```c
struct sockaddr_storage serverAddress; // Structure to hold the server address
socklen_t addr_size;                   // Size of the server address structure

memset(&serverAddress, 0, sizeof(serverAddress)); // Clear the serverAddress structure

// Configure the server address based on the address family
if (family == AF_INET)
{
    struct sockaddr_in *addr_in = (struct sockaddr_in *)&serverAddress;
    addr_in->sin_family = family;                    // Set address family to IPv4
    addr_in->sin_port = htons(port);                 // Set the port number (converted to network byte order)
    inet_pton(family, ipAddress, &(addr_in->sin_addr)); // Convert the IP address string to binary form and store it
    addr_size = sizeof(struct sockaddr_in);
}
else if (family == AF_INET6)
{
    struct sockaddr_in6 *addr_in6 = (struct sockaddr_in6 *)&serverAddress;
    addr_in6->sin6_family = family;                   // Set address family to IPv6
    addr_in6->sin6_port = htons(port);                // Set the port number (converted to network byte order)
    inet_pton(family, ipAddress, &(addr_in6->sin6_addr)); // Convert the IP address string to binary form and store it
    addr_size = sizeof(struct sockaddr_in6);
}
```

Figura 22

- **Envia as informações do cliente:** A função `sendto()` é usada para enviar a estrutura `client_info` para o servidor. Essa estrutura contém o descritor do socket do cliente, o filme escolhido e o índice da citação atual. [figura 23]

```c
// Send the client information to the server
sendto(clientSocket, &info, sizeof(info), 0, (struct sockaddr *)&serverAddress, addr_size);
```

Figura 23

- **Recebe e imprime as citações:** Um loop é usado para receber e imprimir as citações do servidor. A função `recvfrom()` é usada para receber os dados em modo de bloqueio. Cada citação recebida é exibida usando `printf()`.[figura 24]

```c
// Receive and print quotes from the server
for (int i = 0; i < 5; i++)
{
    recvfrom(clientSocket, serverResponse, MAX_SIZE, 0, NULL, NULL); // Receive data from the server (blocking call)
    printf("%s\n", serverResponse);                                  // Print the received quote
}
```

Figura 24

`choose_movie(char *ipVersion, char *ipAddress, int port)`: Esta função permite que o usuário escolha um filme e inicie a comunicação com o servidor. As etapas executadas são:

- **Apresenta um menu de opções:** Um menu com os filmes disponíveis é exibido para o usuário. [figura 25]

```
printf("0 - Sair\n");
printf("1 - Senhor dos aneis\n");
printf("2 - O poderoso chefão\n");
printf("3 - Clube da luta\n");
```

Figura 25

- **Lê a escolha do usuário:** A função `scanf()` é usada para ler a escolha do usuário. [figura 26]

```
scanf("%d", &clientAction)
```

Figura 26

- **Valida a entrada do usuário:** A entrada do usuário é verificada para garantir que seja um número válido dentro do intervalo de opções. Se a entrada for inválida, uma mensagem de erro é exibida. [figura 27]

```
if (clientAction == 0)
{
    break; // Exit the loop if the user chooses to exit
}
else if (clientAction >= 1 && clientAction <= 3) // Validate the user's choice
{
    sa_family_t family = strcmp(ipVersion, "ipv4") == 0 ? AF_INET : AF_INET6; // Determine the address family based on the IP version
```

Figura 27

- **Cria um socket UDP:** A função `create_socket()` é chamada para criar um socket UDP com base na versão do `IP (ipVersion)`. [figura 28]

```
int clientSocket = create_socket(family); // Create a UDP socket
```

Figura 28

- **Inicializa a estrutura client_info:** Uma estrutura `client_info` é inicializada com o descritor do socket, o filme escolhido e o índice da citação inicial (0). [figura 29]

```
client_info info = {clientSocket, clientAction, 0}; // Initialize the client_info structure
```

Figura 29

- **Chama a função connect_socket:** A função `connect_socket()` é chamada para estabelecer a comunicação com o servidor e enviar as informações do cliente. [figura 30]

```
connect_socket(clientSocket, family, ipAddress, port, info); // Connect to the server and receive quotes
```

Figura 30

`main(int argc, char **argv)`: A função `main()` é o ponto de entrada do programa. Ela executa as seguintes etapas:

- **Valida os argumentos da linha de comando:** Verifica se o número de argumentos está correto e extrai a versão do `IP (IPv4 ou IPv6)`, o endereço `IP` do servidor e o número da porta. [figura 31]

```
if (argc ≠ 4)
{
    printf("Número inválido de argumentos\n");
    exit(EXIT_FAILURE); // Exit if the number of arguments is incorrect
}

char *ipVersion = argv[1]; // Get the IP version (e.g., "ipv4" or "ipv6")
char *ipAddress = argv[2]; // Get the server's IP address
int port = atoi(argv[3]);  // Convert the port number to an integer

// Validate the IP version
if (strcmp(ipVersion, "ipv4") ≠ 0 && strcmp(ipVersion, "ipv6") ≠ 0)
{
    printf("Versão de IP inválida\n");
    exit(EXIT_FAILURE); // Exit if the IP version is invalid
}
```

Figura 31

- **Chama a função choose_movie:** A função `choose_movie()` é chamada para iniciar a interação cliente-servidor. [figura 32]

```
choose_movie(ipVersion, ipAddress, port); // Start the client-server interaction
```

Figura 32

Detalhes da Conexão

- `socket()`[figura 21]: Esta função é utilizada para criar um socket. Ela recebe três argumentos: a família de endereços (por exemplo, AF_INET para IPv4), o

tipo de socket (por exemplo, SOCK_DGRAM para UDP) e o protocolo (geralmente 0 para o protocolo padrão). A função retorna um descritor de socket que pode ser usado em chamadas de função subsequentes que operam em sockets.

- `sendto()`[figura 23]: Esta função é usada para enviar dados através do socket para o servidor conectado. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para os dados a serem enviados, o terceiro argumento é o tamanho dos dados, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes enviados ou -1 em caso de erro.
- `recvfrom()`[figura 24]: Esta função é usada para receber dados do servidor conectado através do socket. O primeiro argumento é o descritor do socket, o segundo argumento é um ponteiro para um buffer onde os dados serão armazenados, o terceiro argumento é o tamanho do buffer, e o quarto argumento especifica opções (geralmente 0). A função retorna o número de bytes recebidos ou -1 em caso de erro.
- `inet_pton()`[figura 22]: Esta função converte endereços IP de notação textual para formato binário. O primeiro argumento é a família de endereços (por exemplo, AF_INET para IPv4), o segundo argumento é uma string contendo o endereço IP em notação textual, e o terceiro argumento é um ponteiro para uma estrutura que será preenchida com o endereço IP em formato binário. A função retorna 1 se a conversão for bem-sucedida ou -1 em caso de erro.
- `close()`[figura 24]: Fecha o socket.

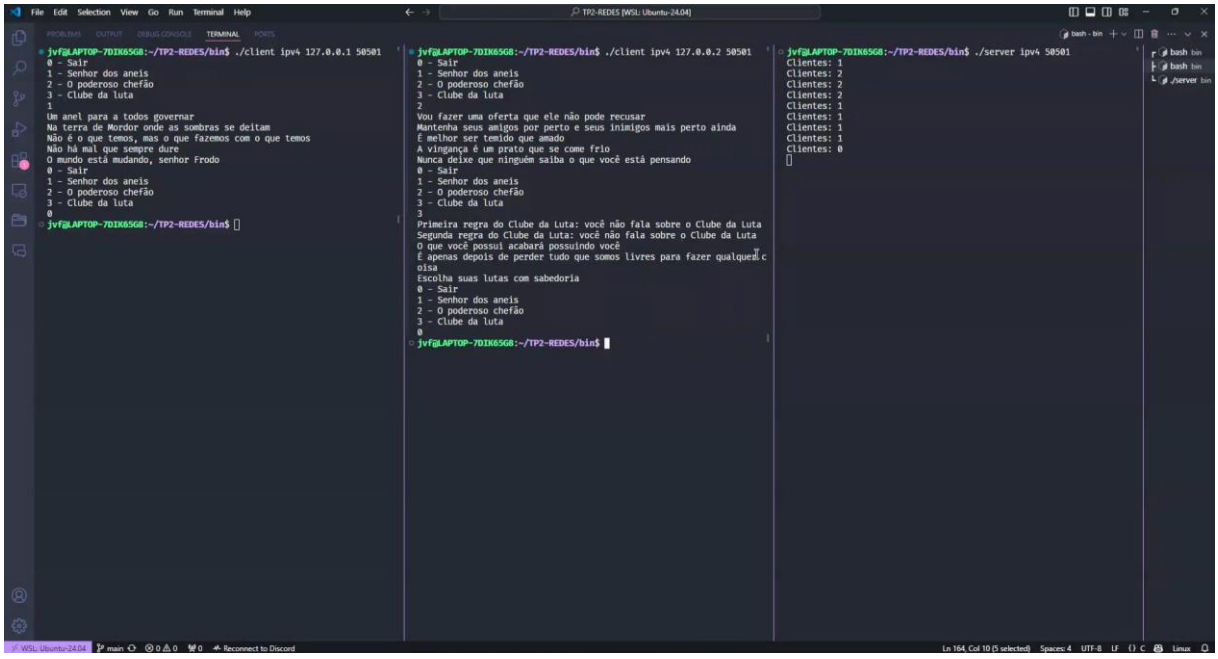
TESTES

IPv4

Para a realização do teste de IPv4 foram utilizados dois clientes simultâneos. Os passos executados em ordem foram:

1. Início cliente 1
2. Cliente 1 visualiza o menu de opções
3. Início cliente 2
4. Cliente 2 visualiza o menu de opções
5. Início servidor
6. Cliente 1 escolhe o filme 1
7. Cliente 1 recebe a frase "Um anel para a todos governar"
8. Servidor mostra: Clientes: 1
9. Cliente 2 escolhe filme 2
10. Cliente 2 recebe a frase 1 "Vou fazer uma oferta que ele não pode recusar"
11. Cliente 1 recebe frase 2 "Na terra de Mordor onde as sombras se deitam"
12. Cliente 2 recebe frase 2 "Mantenha seus amigos por perto e seus inimigos mais perto ainda"
13. Servidor mostra: Clientes: 2
14. Cliente 1 recebe frase 3 "Não é o que temos, mas o que fazemos com o que temos"
15. Cliente 2 recebe frase 3 "É melhor ser temido que amado"
16. Servidor mostra: Clientes: 2
17. Cliente 1 recebe frase 4 "Não há mal que sempre dure"
18. Cliente 2 recebe frase 4 "A vingança é um prato que se come frio"
19. Servidor mostra: Clientes: 2
20. Cliente 1 recebe frase 5 "O mundo está mudando, senhor Frodo"
21. Cliente 1 visualiza o menu de opções
22. Cliente 1 escolhe a opção sair
23. Cliente 2 recebe frase 5 "Nunca deixe que ninguém saiba o que você está pensando"
24. Cliente 2 visualiza o menu de opções
25. Cliente 2 escolhe filme 3

26. Cliente 2 recebe frase 1 "Primeira regra do Clube da Luta: você não fala sobre o Clube da Luta"
27. Servidor mostra: Clientes: 1
28. Cliente 2 recebe frase 2 "Segunda regra do Clube da Luta: você não fala sobre o Clube da Luta"
29. Servidor mostra: Clientes: 1
30. Cliente 2 recebe frase 3 "O que você possui acabará possuindo você"
31. Servidor mostra: Clientes: 1
32. Cliente 2 recebe frase 4 "É apenas depois de perder tudo que somos livres para fazer qualquer coisa"
33. Servidor mostra: Clientes: 1
34. Cliente 2 recebe frase 5 "Escolha suas lutas com sabedoria"
35. Cliente 2 escolhe a opção sair
36. Servidor mostra: Clientes: 0

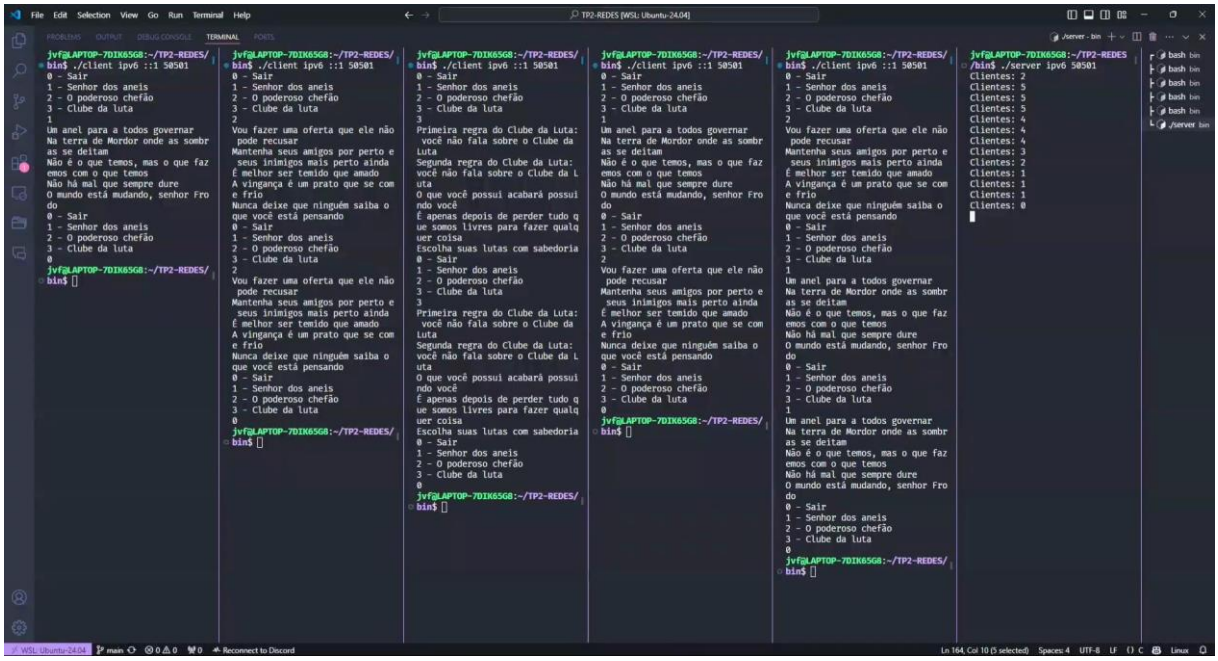


Também é possível ver a execução do programa através deste link: [Exemplo IPv4]

IPv6

Para a realização do teste de IPv6 foram utilizados cinco clientes simultâneos. Os passos executados de forma simplificada foram:

1. *Início* cliente 1
2. *Início* cliente 2
3. *Início* cliente 3
4. *Início* cliente 4
5. *Início* cliente 5
6. *Início* servidor
7. Cliente 1 escolhe filme 1 e recebe as frases do filme
8. Cliente 2 escolhe filme 2 e recebe as frases do filme
9. Cliente 3 escolhe filme 3 e recebe as frases do filme
10. Cliente 4 escolhe filme 1 e recebe as frases do filme
11. Cliente 5 escolhe filme 2 e recebe as frases do filme
12. Cliente 1 escolhe a opção sair
13. Cliente 2 escolhe filme 2 novamente e recebe as frases do filme
14. Cliente 3 escolhe filme 3 novamente e recebe as frases do filme
15. Cliente 4 escolhe filme 2 e recebe as frases do filme
16. Cliente 5 escolhe filme 1 e recebe as frases do filme
17. Cliente 2 escolhe a opção sair
18. Cliente 3 escolhe a opção sair
19. *Cliente 4 escolhe a o*pção sair
20. Cliente 5 escolhe filme 1 novamente e recebe as frases do filme



Também é possível ver a execução do programa através deste link: [Exemplo IPv6]