

Trabalho Prático 1 - Documentação

Ana Beatriz Soares Cardoso - 2022108170

Introdução

O processo para a implementação do Trabalho Prático “JokenBoom” trouxe a oportunidade de aplicar conceitos fundamentais sobre comunicação cliente-servidor utilizando sockets POSIX. O objetivo era criar um sistema que simulasse um duelo entre cliente (humano) e o servidor (máquina), seguindo um protocolo de comunicação bem definido por regras e estruturas de mensagens detalhadas que organizam como cliente e servidor interagem durante o jogo. No entanto, como em toda implementação de um projeto, surgiram desafios que exigiram atenção e principalmente planejamento, desde a compreensão inicial dos requisitos até a implementação prática do sistema. Este documento tem como objetivo relatar dificuldades encontradas e soluções adotadas para superar tais obstáculos.

Compreensão inicial dos requisitos

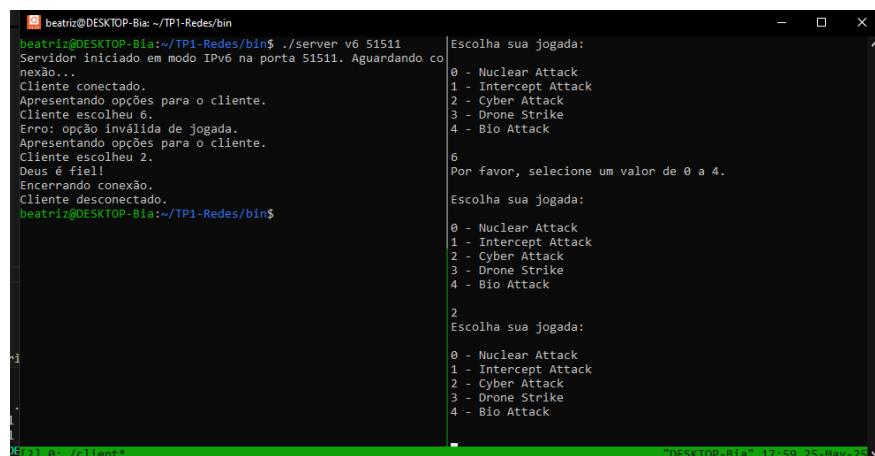
No início do desenvolvimento do trabalho, surgiram algumas dificuldades para compreender plenamente os requisitos funcionais. A exigência de o sistema suportar tanto IPv4 quanto IPv6, associada ao pouco conhecimento prévio sobre sockets e a comunicação cliente-servidor, tornou o início do projeto desafiador. Para superar essas dificuldades, foi realizada uma revisão das aulas sobre sockets sob consulta dos vídeos disponibilizados pelo professor Ítalo Cunha, que abordavam aplicações semelhantes às exigidas no Jokenboom. Esses materiais foram fundamentais para entender como configurar ambos os protocolos IP e estabelecer a comunicação necessária.

Outro desafio enfrentado foi a adaptação do desenvolvimento, originalmente projetado para rodar em Linux, a um ambiente com o sistema operacional Windows. Já fazendo o uso do Visual Studio Code configurado com um terminal Linux (Ubuntu), faltava um ferramenta que permitisse rodar simultaneamente o servidor e o cliente em uma tela dividida. O Tmux se mostrou útil garantindo a possibilidade de testar e depurar o código em um ambiente mais próximo das condições exigidas pelo trabalho, trazendo maior eficiência ao processo de desenvolvimento.

Implementação prática do sistema

Durante a implementação da lógica do jogo, surgiram algumas dificuldades básicas devido à falta de prática recente em linguagem C. Essas dificuldades incluíram desde a necessidade de relembrar conceitos até a elaboração de uma lógica funcional para a implementação. Após compreender o fluxo de dados e as trocas de mensagens entre servidor e cliente, utilizando a estrutura de dados fornecida, o primeiro tratamento de erros foi iniciado: verificar se o cliente escolhia uma opção inválida entre as jogadas disponíveis.

A lógica principal do jogo foi implementada dentro de um laço `while(1)`. Após o recebimento da mensagem `MSG_RESPONSE` do cliente, um bloco `if-else` foi utilizado para validar se a jogada estava dentro do intervalo `[0 - 4]`. Caso a jogada fosse inválida, o bloco `if` fazia uso das mensagens de erro entre cliente e servidor; caso fosse válida, o bloco `else` dava continuidade ao jogo. No entanto, um problema foi identificado: ocorria uma impressão a mais da tabela de ações no terminal do cliente mesmo após a ação ser válida, conforme ilustrado na Figura 1.



```
beatriz@DESKTOP-Bia: ~/TP1-Redes/bin
beatriz@DESKTOP-Bia:~/TP1-Redes/bin$ ./server v6 51511
Servidor iniciado em modo IPV6 na porta 51511. Aguardando co
nexão...
Cliente conectado.
Apresentando opções para o cliente.
Cliente escolheu 6.
Erro: opção inválida de jogada.
Apresentando opções para o cliente.
Cliente escolheu 2.
Deus é fiel!
Encerrando conexão.
Cliente desconectado.
beatriz@DESKTOP-Bia:~/TP1-Redes/bin$

Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack

6
Por favor, selecione um valor de 0 a 4.

Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack

2
Escolha sua jogada:
0 - Nuclear Attack
1 - Intercept Attack
2 - Cyber Attack
3 - Drone Strike
4 - Bio Attack
```

Figura 1: Erro na impressão das mensagens no terminal do cliente.

Para resolver esse problema, foi criada uma variável auxiliar chamada `auxValid`, do tipo `int`, inicializada com o valor 1 e declarada fora do laço `while`. Dentro do laço, o trecho de código responsável por enviar a tabela de ações ao cliente foi colocado dentro de um `if` onde a condição dependia de `auxValid` ser verdadeiro. Nos blocos `if-else` utilizados para o tratamento de erros, a variável `auxValid` sofreu mudanças de estado: no caso de uma jogada inválida (`if`), `auxValid` recebia o valor 1, permitindo que a tabela fosse enviada novamente ao cliente quando a revalidação do `if` dependente de `auxValid` fosse feita; no caso de uma jogada válida (`else`), `auxValid` recebia o valor 0, impedindo novas impressões desnecessárias da tabela.

Essa abordagem garantiu que a tabela de ações fosse exibida apenas na primeira interação ou quando o cliente inserisse uma jogada inválida, corrigindo o erro de impressão extra.

Outras aplicações para a lógica do jogo ocorreram sem problemas.

Reformulação de trechos

Durante o processo de entendimento de como funcionava a programação de sockets, tomando como base essencialmente as vídeo-aulas do professor Ítalo Cunha, partes do código para a configuração do endereço de rede e para a configuração do endereço do servidor para uma determinada rede (funções implementadas no arquivo *common_module.c*) ficaram muito semelhantes às vistas na vídeo-aula. Por esse motivo, foram pesquisadas e implementadas outras abordagens para realizar essas mesmas funcionalidades, a fim de diversificar e principalmente aprender.

Para a configuração do endereço de redes, função *int configura_addr()*, ao invés de fazer manualmente a conversão para cada tipo de IP (v4 ou v6) utilizando *inet_pton()*, uma abordagem encontrada foi utilizar a função *getaddrinfo()*, também disponível na POXIS. Essa função lida tanto com endereços IPv4 quanto IPv6 de forma automática, e permite obter uma lista de endereços associados a um nome ou a uma string de endereço, junto com informações adicionais como tipo de socket e protocolo, simplificando a configuração de sockets para diferentes protocolos de rede.

```
11 // Função para configurar o endereço de rede (v4 ou v6) em uma estrutura do tipo sockaddre_storage
12 // Função para configurar o endereço de rede (v4 ou v6) em uma estrutura do tipo sockaddre_storage
13 /* Trecho inspirado na aula do Prof. Ítalo Cunha */
14 int configura_addr(const char *str_addr, const char *porta_str, struct sockaddr_storage *server_addr){
15     uint16_t porta_int = htons((uint16_t)atoi(porta_str)); // Converte a string da porta para um número inteiro
16     // htons converte o número inteiro da porta para big-endian (usado em rede)
17     struct in_addr endereco_v4;
18     struct in6_addr endereco_v6;
19     if(inet_pton(AF_INET, str_addr, &endereco_v4)){ // inet_pton converte str_addr para binário, e testa se é v4
20         struct sockaddr_in *addr_v4 = (struct sockaddr_in *)server_addr; // conversão de tipo, para struct sockaddr_in
21         addr_v4->sin_family = AF_INET; // Definição da família
22         addr_v4->sin_port = porta_int; // sin_port armazena a porta em formato big-endian
23         addr_v4->sin_addr = endereco_v4; // sin_addr armazena o endereço em binário
24         return 0;
25     }
26     if(inet_pton(AF_INET6, str_addr, &endereco_v6)){
27         struct sockaddr_in6 *addr_v6 = (struct sockaddr_in6 *)server_addr;
28         addr_v6->sin6_family = AF_INET6;
29         addr_v6->sin6_port = porta_int;
30         memcpy(&(addr_v6->sin6_addr), &endereco_v6, sizeof(endereco_v6));
31         /* memcpy foi utilizado para copiar os 16 bytes do endereço IPv6 (endereco_v6) para o campo sin6_addr,
32         já que sin6_addr é um array de bytes e endereco_v6 é uma estrutura in6_addr. Isso garante que todos
33         os bytes do endereço sejam copiados da maneira correta */
34         return 0;
35     }
36     return -1;
37 }
38
```

Figura 2: função *configura_addr()*, abordagem anterior: *inet_pton()*.

```

10
11 // Função para configurar o endereço de rede (v4 ou v6) em uma estrutura do tipo sockaddr_storage
12 int configura_addr(const char *str_addr, const char *porta_str, struct sockaddr_storage *server_addr){
13
14     struct addrinfo tipo_protocolo, *resultados_gerados;
15     memset(&tipo_protocolo, 0, sizeof(tipo_protocolo));
16     tipo_protocolo.ai_family = AF_UNSPEC; // Campo ai_family como AF_UNSPEC (aceita IPv4 e IPv6)
17     tipo_protocolo.ai_socktype = SOCK_STREAM; // Tipo de socket (TCP)
18
19     getaddrinfo(str_addr, porta_str, &tipo_protocolo, &resultados_gerados); // getaddrinfo resolve str_addr e porta_str co
20     // e preenche a lista de resultados em resultados_gerados.
21
22     memcpy(server_addr, resultados_gerados->ai_addr, resultados_gerados->ai_addrlen); // Cópia do endereço resolvido
23     freeaddrinfo(resultados_gerados); // Libera a memória alocada por getaddrinfo
24     return 0;
25 }
26

```

Figura 3: função `configura_addr()`, abordagem atual: `getaddrinfo()`.

O trecho da função `int inicializar_addr_server()` foi mantido semelhante ao na vídeo-aula, a fim de demonstrar uma abordagem manual da configuração do endereço do servidor ao determinado tipo de redes escolhido.

Considerações finais

A execução deste trabalho prático permitiu o aprendizado de conceitos essenciais de programação de sockets e comunicação cliente-servidor na prática, aplicando conhecimentos sobre a criação, configuração e utilização de sockets em IPv4 e em IPv6, com foco no envio e recepção de dados.

Mesmo com dificuldades iniciais causadas pela falta de prática na linguagem e pelo ainda pouco entendimento sobre o assunto, o desenvolvimento progrediu de maneira fluida e o trabalho cumpriu os objetivos propostos. A experiência adquirida será útil em futuros projetos e aplicações em redes e sistemas.

Referências

getaddrinfo. Disponível em: < <https://pubs.opengroup.org/onlinepubs/009619199/getad.htm> > Acesso em: 24 mai 25

Introdução à Programação em Redes. por Ítalo Cunha. Disponível em: < https://www.youtube.com/watch?v=tJ3qNtv0HVs&list=PLyrH0CFXIM5Wzmbv-lC-qvoBejsa803Qk&ab_channel=%C3%8DtaloCunha > Acesso em: 01 mai 25