Cliente/Servidor Seguro com Compactação e Criptografia TLS

Autores: Vitor Jordao C. Briglia, Hendrick Silva Ferreira, Guilherme Miranda de Araújo

Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)
Boa Vista – RR – Brasil

vitorjordao8762@gmail.com, h_ferreira@gmail.com,
guimiranda2003@gmail.com

Abstract: *This paper presents an analysis and constraints of a course project focused on developing a network communication system with data compression. Theoretical and practical aspects of TCP sockets, Zlib specifications, and multi-client management via processes were explored.*

Resumo: *Este trabalho apresenta uma análise e reestruturação do projeto de disciplina voltado ao desenvolvimento de um sistema de comunicação de rede com compactação de dados. Foram explorados aspectos teóricos e práticos no uso de sockets TCP, compressão com Zlib e gerenciamento de múltiplos clientes via processos.*

1. Introdução

O protocolo Telnet é uma tecnologia de comunicação remota que permite a execução de comandos em terminais distantes. Utilizado inicialmente por militares, passou a ser explorado amplamente na Internet a partir dos anos 1970. Neste projeto, sua arquitetura foi estudada como base para um sistema mais moderno e seguro, adaptado com compressão e execução multiusuário.

2. Comunicação Cliente-Servidor com TCP

As conexões TCP/IP podem ser comparadas a uma ligação telefônica: alguém precisa iniciar a chamada, enquanto do outro lado, alguém deve estar disponível para atender.

Nesse contexto, o endereço IP funciona como o número do telefone, e a porta representa uma espécie de ramal — aquele ponto específico que será utilizado para a comunicação após o contato inicial.

Em uma conexão TCP/IP, chamamos de “Cliente” o dispositivo ou computador que faz a chamada, ou seja, que inicia a comunicação. Já o “Servidor” é quem está do outro lado, aguardando ativamente por essas conexões. Para que a comunicação aconteça, o cliente precisa saber o endereço IP do servidor e também qual porta utilizar para troca de dados.

Uma vez estabelecida a conexão entre cliente e servidor por meio de uma porta TCP, os dados podem trafegar livremente nos dois sentidos — como acontece com outros tipos de conexões físicas (por exemplo, portas seriais ou paralelas), mas, nesse caso, por meio da rede. A conexão permanece ativa até que um dos lados — cliente ou servidor — decida encerrá-la, como se desligasse o telefone.

Uma das grandes vantagens do protocolo TCP/IP está na confiabilidade: os dados transmitidos passam por verificações automáticas de integridade feitas pelos drivers de baixo nível, o que garante que nenhuma informação chegue corrompida ou com erro.

No nosso projeto, utilizamos o protocolo TCP para viabilizar a comunicação entre os módulos “Cliente” e “Servidor”. Com o apoio da tecnologia de Sockets — que permite a troca de informações entre processos, seja na mesma máquina ou em diferentes dispositivos da rede — conseguimos distribuir o processamento de forma eficiente e acessar os dados de maneira centralizada. Neste caso, tanto o cliente quanto o servidor estão sendo executados localmente, via localhost.

3. Uso do Fork

A função fork foi usada no servidor para habilitar a execução simultânea de múltiplos clientes. A replicação de processos permitiu tratar cada cliente de forma isolada, melhorando a escalabilidade e robustez do sistema.

3.1 Processos do tipo Daemon

Foi discutida a criação de processos do tipo daemon, que rodam em segundo plano e são independentes do terminal. Esse conceito reforça a ideia de serviços persistentes, úteis em sistemas operacionais modernos.

4. Desenvolvimento da Solução

Durante a implementação, foram criados dois módulos principais: cliente e servidor. O cliente estabelece conexão e envia dados; o servidor os recebe, processa e responde. Para testes, utilizou-se a estratégia de Echo Server com suporte a múltiplos clientes via fork.

4.1. Codigo Cliente

Aqui no código do “Cliente” foi incluído as bibliotecas necessárias para usarmos algumas funções, como a do Socket por exemplo. Conseguimos com sucesso fazer a criação do Socket no cliente e também fazer a verificação. A configuração com o “Servidor” também foi realizada com sucesso , assim como a verificação da conexão também. Outra coisa que conseguimos definir no nosso código foi a porta de acesso ao servidor.

```c

int main(){
    int clientSocket, ret;
    struct sockaddr_in serverAddr;
    char buffer[1024];

    // Criação do socket
    clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if(clientSocket < 0){
        printf("[-] Erro na criação do socket.\n");
        exit(1);
    }
    printf("[+] Socket do cliente criado!\n");

    // Configuração do endereço do servidor
    memset(&serverAddr, '\0', sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT);
    serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Tentativa de conexão
    ret = connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    if(ret < 0){
        printf("[-] Erro na conexão.\n");
        exit(1);
    }
    printf("[+] Conectado ao servidor.\n");

    while(1){
        bzero(buffer, sizeof(buffer));
        printf("Client:\t");
        scanf("%s", buffer);  // cuidado: não lê espaços!

        send(clientSocket, buffer, strlen(buffer), 0);

        if(strcmp(buffer, ":exit") == 0){
            close(clientSocket);
            printf("[-] Conexão encerrada com o servidor.\n");
            break;
        }

        bzero(buffer, sizeof(buffer));
        if(recv(clientSocket, buffer, sizeof(buffer), 0) < 0){
            printf("[-] Erro ao receber dados.\n");
        }else{
            printf("Server:\t%s\n", buffer);
        }
    }

    return 0;
}


```

Figura 1. Código Cliente

4.2. Codigo Servidor

Já no código do “Servidor” encontramos alguns impasses que logo foram resolvidos após algumas pesquisas. Dessa forma, conseguimos declarar as variáveis do Socket e Buffer.Definimos e configuramos os parâmetros de endereçamento pro “Cliente” e fizemos também a associação do Socket com o localhost. E como foi dito, usamos o fork para conseguir produzir um “Servidor” que fosse “Multicliente”, fazendo assim cada um ter seu IP de acesso.

```
15
16      int newSocket;
17      struct sockaddr_in newAddr;
18      socklen_t addr_size;
19
20      char buffer[1024];
21      pid_t childpid;
22
23      // Criação do socket
24      sockfd = socket(AF_INET, SOCK_STREAM, 0);
25      if(sockfd < 0){
26          printf("[-] Erro na criação do socket.\n");
27          exit(1);
28      }
29      printf("[+] Socket do servidor criado!\n");
30
31      // Permitir reuso da porta
32      int opt = 1;
33      setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
34
35      // Configuração do endereço
36      memset(&serverAddr, '\0', sizeof(serverAddr));
37      serverAddr.sin_family = AF_INET;
38      serverAddr.sin_port = htons(PORT);
39      serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
40
41      // Associação do socket com o endereço local
42      ret = bind(sockfd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
43      if(ret < 0){
44          printf("[-] Erro na vinculação. \n");
45          exit(1);
46      }
47      printf("[+] Vinculado à porta %d\n", PORT);
48
49      if(listen(sockfd, 10) == 0){
50          printf("[+] Aguardando conexão...\n");
51      }else{
52          printf("[-] Erro ao escutar.\n");
53      }
54
55      while(1){
56          addr_size = sizeof(newAddr);
57          newSocket = accept(sockfd, (struct sockaddr*)&newAddr, &addr_size);
58          if(newSocket < 0){
59              exit(1);
60          }
61          printf("Conexão aceita com %s:%d\n", inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));
62
63          if((childpid = fork()) == 0){
64              close(sockfd);
65
66              while(1){
67                  bzero(buffer, sizeof(buffer));
68                  recv(newSocket, buffer, sizeof(buffer), 0);
69                  if(strcmp(buffer, ":exit") == 0){
70                      printf("Desconectado de %s:%d\n", inet_ntoa(newAddr.sin_addr), ntohs(newAddr.sin_port));
71                      break;
72                  }else{
```

Figura 2. Código Servidor

5. Limitações e Desafios

Algumas funcionalidades planejadas não foram concluídas. Entre elas: suporte aos comandos --log, --host e --compress, comunicação com o shell via pipe, e compressão de dados com Zlib. Os desafios envolveram principalmente dificuldade na manipulação de arquivos de log e no uso das bibliotecas de compressão.

6. Considerações Finais

A implementação da comunicação entre cliente e servidor utilizando o protocolo TCP/IP mostrou-se essencial para garantir uma troca de dados segura, confiável e eficiente. Ao adotar esse protocolo em conjunto com o uso de sockets, conseguimos estabelecer uma base sólida para o funcionamento da nossa aplicação, permitindo que os processos se comuniquem de forma estável, mesmo que estejam em máquinas diferentes ou em uma mesma rede local.

Durante o desenvolvimento, ficou evidente como esses recursos facilitam a integração entre diferentes partes do sistema, distribuindo tarefas de maneira inteligente e promovendo maior aproveitamento dos recursos disponíveis. Além disso, o uso do localhost como ambiente de testes nos proporciona praticidade e controle durante a fase de desenvolvimento, sem comprometer a estrutura do projeto.

Essa experiência reforça a importância de entender bem os fundamentos das redes de computadores e dos protocolos de comunicação, já que eles estão diretamente ligados ao sucesso de sistemas distribuídos modernos. Em suma, o uso do TCP/IP com sockets não apenas cumpriu seu papel técnico, mas também ampliou nossa compreensão sobre os desafios e soluções envolvidas na construção de aplicações conectadas em rede.

Referências

- SILVA, Cristiano. fork, exec, e daemon. Embarcados, 2021 .<https://embarcados.com.br/socket-tcp/#Biblioteca>

- GEEKSFORGEEKS. Handling multiple clients with sockets, 2022. <https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/>- INGALLS, Robert. Sockets Tutorial, 1998.

- IDIOT DEVELOPER. TCP Client Server in C. YouTube, 2021. <https://www.youtube.com/watch?v=io2G2yW1Qk8>