Compressed Network Communication

Kelvin A. Ferreira[1], Leonardo C. de Matos Silva[2], Victor R. de Souza Teixeira[3]

Docente: Herbert Rocha

Universidade Federal de Roraima (UFRR) – Departamento de Ciência da Computação

kelvin.ufrr@gmail.com, leonardo.carvalho623@gmail.com,
vroberto1234567890@gmail.com

Abstract. *This article describes the details of creating, configuring and maintaining a communication network through a telnet client and server through a TCP socket with the communication between them compressed.*

Resumo. *Este artigo descreve os detalhes sobre a criação, configuração e manutenção de uma rede de comunicação através de um cliente e um servidor de telnet por meio de um socket TCP com a comunicação entre eles compactada.*

1. Introdução

O objetivo do projeto realizado é mostrar como funciona a comunicação via TCP e estabelecer uma conexão entre um cliente e um servidor. Entregar comandos ao cliente e fazer o servidor retornar os mesmos. Implementar o código em C fazendo a chamada de funções específicas. Teremos um processo de cliente, um processo de servidor e um processo de shell. Os dois primeiros são conectados via conexão TCP, e os dois últimos são conectados via pipes, como mostra a figura abaixo.

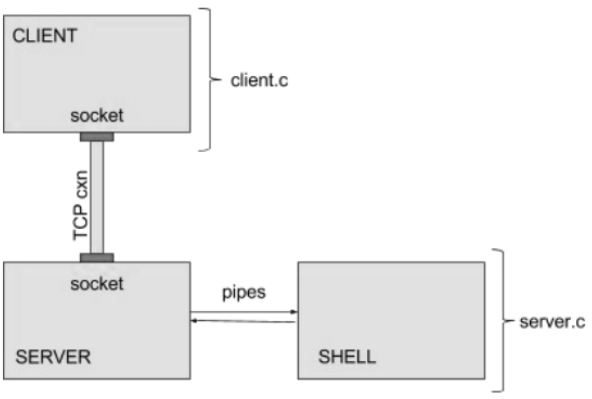

Figura 1. Estrutura do projeto

2. Desenvolvimento

2.1. Implementação Superficial

Para desenvolver este projeto tínhamos as opções de linguagens C, C++ e Rust. Dentre estas foi escolhida a linguagem de programação C, por ser a com melhor afinidade dos integrantes do grupo e a com maior referência na web.

Primeiro foi estudado em qual sistema operacional seria implementado o código, a priori seria baseado nas bibliotecas do Linux, mas após alguns dias de pesquisa foram escritas algumas linhas baseadas no Windows, mas sem progresso no C, em Python foi escrito um código funcional que atende a grande dos requisitos solicitados, todavia seguindo a mesma proposta das bibliotecas do Windows a ideia foi deixada de lado e focamos no projeto principal, escrito no Linux.

Nos programas, temos Servidor e Cliente, ambos implementados em C. Na sua estrutura temos primeiramente declarado quais bibliotecas serão utilizadas, todas presentes no pacote do GCC, com exceção da biblioteca de compactação ZLib.

Foi declarado um local host, endereço de IP, e uma porta de acesso, criando o endereço (IPV4) “127.0.0.1:5432”. No projeto o servidor e o cliente estarão na mesma máquina, então precisaria ser declarado o endereço de IP e a mesma porta para ambos, o local host foi declarado como ‘127.0.0.1’ pois sendo este o endereço local, onde que é digitado no terminal, retorna para você novamente, a porta precisaria apenas ser qualquer valor acima de 5000 para garantir que não haveria conflito com outras portas já usadas por padrão pelo sistema.

Definimos um tamanho para o buffer pensando na lógica de um heap, nas primeiras implementações ele estava o tempo todo estourando a memória por causa do protocolo das mensagens que acabavam por ser grande demais. Chegamos a este valor de 16384 multiplicando 16 bites por 1024 KB.

A função bind() é responsável por escolher qual porta usar, e por qual socket estamos usando, Se o bind me retornar 0, tá tudo certo. Se ele retornar qualquer valor acima de 0, deu erro, quais os possíveis erros. Ele tentou abrir uma porta em uso, ou uma porta já aberta, portas estas que não podem ser utilizadas. Se for abaixo de zero, o cliente está com a porta fechada ou nem abriu a porta.

2.2. Ferramentas Utilizadas

Para desenvolver o projeto utilizamos as seguintes ferramentas: O aplicativo WhatsApp e o Discord para nos comunicarmos durante o desenvolvimento; O software Canva para produzir a apresentação em PDF em equipe; A ferramenta Google Docs para

elaboração do presente artigo; O site whimsical.com para organizar e dividir as atividades entre o grupo ; A IDE Visual Studio Code para implementar os nossos códigos; E o gerenciador de repositórios GIT e GitHub para fazer o versionamento do projeto.

3. Socket TCP

O soquete estabelece a conexão entre o servidor e o cliente através do IP e das portas que foram declaradas, Ele define o protocolo de rede e é nele que declaramos o número de flags que vamos utilizar, ou seja, dentro dele que é chamada a função e declarado o nosso número máximo de clientes acessando o servidor simultaneamente.

As funções que dependem dessa chamada e foram utilizadas em nosso código foram: socket() cria o socket que vai fazer a conexão; bind() faz a conexão entre o socket e a porta; listen() Chamada do socket pela porta, dentro dele é declarado o número limite da fila de conexões, essa fila é organizada como uma fila FIFO de conexões pendentes; accept() quando efetuada uma ligação, ele quem aceita e estabelece a conexão; connect() declarada no lado do cliente, estabelece a comunicação com o servidor; send() - caso conectado, envia mensagens ao socket; recv() - recebe mensagem por meio do socket, e repassa através de um ponteiro para outra função; close() – encerra o descritor de arquivos.

4. Pipe / Fork

O pipe é a ligação entre o Servidor e o Shell do Linux, nele é chamada a função Fork() para conseguir escrever e ler dentro do pipe.

A função Fork() é utilizada para criar um processo filho, esse processo filho é uma cópia do processo original.

5. Compressão de Dados (zlib)

Para comprimir os dados foi utilizada a biblioteca zlib. Ela permite a compressão e descompressão do buffer enviado a elas.

Zlib – é uma biblioteca multiplataforma de compressão de dados, criada para popularizar o algoritmo deflate. Basicamente, para compactar, temos que informar o ponteiro do buffer de entrada e seu tamanho e chamar a função deflate() tantas vezes quantas forem necessárias para obter todos os bytes compactados num buffer de saída. Para descompactar o processo é inverso: Fornecemos o buffer de entrada, com os dados

compactados, e chamamos inflate() tantas vezes quantas forem necessárias para obtermos, no buffer de saída, os dados descompactados.

Atualmente, zlib é um padrão de fato para compressão de dados em documentos portáveis. Centenas de aplicações para sistemas operacionais Unix e assemelhados (como o Linux) usam-na e seu uso é crescente em outros sistemas (como o Microsoft Windows e o Palm OS).

6. Servidor

O servidor aceita a conexão com os clientes via socket. Onde temos definição de algumas variáveis para auxiliar na inicialização, sendo uma variável booleana que representa o estado da inicialização do servidor, uma variável do tipo inteiro que recebe o resultado das funções necessárias para a configuração, uma variável do tipo inteiro para habilitar o reuso da porta caso o servidor precise reiniciar e uma estrutura sockaddr_in que é usada para configurar o servidor para se comunicar através da rede.

E também as funções para efetuar o contato com o cliente: para quando acontece erro ao conectar-se com o cliente, para aguardar conexão, aceitar conexão, e encerrar conexão.

7. Cliente

No programa Client definimos algumas variáveis para auxiliar na comunicação com o servidor, sendo uma variável booleana que representa o estado da parametrização do cliente, uma variável do tipo inteiro que recebe o resultado das funções necessárias para a configuração, uma estrutura sockaddr_in que é usada para configurar o servidor no qual será conectado, e duas variáveis de quantidade de dados enviados e recebidos.

8. Problemas Enfrentados

No aplicativo tivemos alguns problemas de implementação como a utilização do comando --Log, ele deveria no código do Cliente manter um registro dos dados enviados pelo soquete, no nosso caso o comando não chama o soquete, ele fica estático.

Outro problema enfrentado foi com o comando --Compress, com o uso da biblioteca da Zlib tentamos utilizar comandos para a compressão dos comandos enviados do Cliente para o Servidor e do Servidor para o Cliente, em priori utilizamos a função Deflate, ela possibilita a compressão de dados só que de arquivos, no nosso caso essa função não se adequava ao código, partimos para a utilização da função Compress para comprimir e

Uncompress para descomprimir, essas funções tratam a compressão de dados em um Buffer, quando foram implementadas no nosso código elas não compactavam as mensagens, talvez por falta de tempo hábil ou por desatenção esses foram os problemas enfrentados no código.

9. Referências

BIBLIOTECA ZLIB, **Documentação** da biblioteca **zlib**. Disponível em: <http://zlib.net/>. Acesso em: 17 jul. 2022.

TERMINAL ROOT, Como utilizar a biblioteca **zlib** com **C**. Disponível em: <https://terminalroot.com.br/2021/06/como-utilizar-a-biblioteca-zlib-com-cpp.html>. Acesso em: 18 jul. 2022.

EMBARCADOS, Biblioteca de **Socket TCP**. Disponível em: <https://embarcados.com.br/socket-tcp/#Biblioteca>. Acesso em: 20 jul. 2022.

PAULO SERGIO ALMEIDA, Cliente-servidor com **Sockets TCP.** Disponível em: <https://gsd.di.uminho.pt/teaching/5306O2/2006/sockets.pdf>. Acesso em: 20 jul. 2022.