Compressed Network Communication

Eduardo H. Machado[1], Rosialdo Q. Vicente[1], Venícius J. Oliveira[1]

[1] Universidade Federal de Roraima (UFRR)
Campus Paricarana – Boa Vista – RR – Brazil

edu.hen.fm@gmail.com, rosialdovidinho3@gmail.com,
veniciusjoliveira@gmail.com

Abstract. *This report describes a CNC (Compressed Network Communications) implementation. It explains the project management, building tools, and the concepts needed while developing it. It also goes about problems, upgrades, and limitations that appeared during its conception.*

Resumo. *Este relatório descreve a implementação de uma CNC (Compressed Network Communications). Ele trata da organização do projeto, ferramentas utilizadas e conceitos necessários no seu desenvolvimento. Além disso, também discute os problemas, melhorias e limitações que surgiram ao longo da sua concepção.*

1. Visão Geral

Este projeto visava a implementação de uma CNC (Compressed Network Communication), que funciona de uma forma similar a um chat. Comandos são enviados pelo cliente ao servidor, que os executa por meio de um pipe, e envia o resultado ao cliente. Cliente e servidor se comunicam por meio de um socket TCP/IP, e os dados passados entre eles é comprimido.

Para implementar o projeto, foram divididas funções entre os membros para melhor andamento do projeto, dado que o prazo foi curto. Todo o código foi escrito em C, desenvolvido e testado em ambientes Linux e seu versionamento se encontra no GitHub.

2. Organização do Projeto

2.1. Divisão das Tarefas

O trabalho foi dividido da seguinte maneira: Venícius Jacob ficou responsável pela implementação do socket que interliga o cliente e o servidor; Eduardo Machado realizou a compressão dos dados usando a biblioteca zlib e implementou as funções do cliente e do servidor; e Rosialdo Vicente desempenhou o papel de conectar o terminal ao servidor, por meio do pipe, para executar comandos.

2.2. Ferramentas Utilizadas

Para o andamento do projeto foram utilizadas as seguintes ferramentas: o aplicativo de mensagens WhatsApp e o organizador de projetos Trello, para a comunicação da equipe; a ferramenta de edição de texto Google Docs, que permitiu a elaboração do presente relatório concorrentemente, com todos os membros da equipe; na edição do código usamos a IDE Visual Studio Code e o gerenciador de repositórios Git, GitHub.

3. Socket TCP

Socket é o que permite que dois processos se comuniquem em uma mesma máquina, ou em máquinas diferentes por meio de uma rede. Isso possibilita uma comunicação bidirecional, ou seja, que envia e recebe mensagens. Analogamente a uma ligação de telefone, você disca um número, e quando a conexão é estabelecida, começa a tocar o telefone da outra pessoa; o mesmo acontece para o socket. No entanto, ao invés de discar o número, são usados o IP e a porta para estabelecer a comunicação. O protocolo TCP é um serviço que garante a entrega e a integridade dos pacotes, de forma ordenada, rodando sobre o protocolo IP. O TCP permite conexões entre processos em máquinas distintas, podendo atribuir funções diferentes para cada máquina.

As principais syscalls utilizadas em relação ao socket são: socket() - cria um socket e retorna o descritor de arquivo; bind() - associa o socket a um endereço socket e uma porta; listen() - entra no modo escuta, aguardando conexões; accept() - quando requisitada uma conexão, aceita e estabelece a conexão; connect() - utilizado no lado do cliente, estabelece a comunicação; send() - caso conectado, transmite mensagens ao socket; recv() - recebe mensagem por meio do socket; close() - fecha o descritor de arquivos.

4. Pipe

Para a implementação do pipe, foi realizada a chamada de um fork(), possibilitando que fossem realizadas duas chamadas do main(). A primeira chamada faz com que consigamos escrever no pipe e na segunda chamada podemos realizar a leitura do pipe. Isso fez com que fosse possível realizarmos a comunicação entre o servidor e o shell, essa ligação aconteceu a partir da função execl(). Ela faz o código ser capaz de executar comandos do shell. Após essa comunicação, focamos em implementar uma forma de conseguir resgatar o conteúdo do output deixado por pelo execl(), que foi por meio do dup2(). Por fim, o resultado é retornado ao servidor.

5. Compressão de Dados (zlib)

Para comprimir os dados foi utilizada a biblioteca zlib,ou mais especificamente as funções compress() e uncompress(). Elas permitiram a compressão e descompressão do buffer enviado a elas, respectivamente.

Essas funções foram usadas ao invés do inflate() e deflate() porque permitiam operar diretamente sobre um vetor de caracteres passado como buffer, ao invés de ter que usar um arquivo externo.

6. Sinais

Há dois sinais que devem ser utilizados nesse projeto. Um é o SIGINT (^C), que seria recebido pelo cliente, e sua execução consiste em interromper o shell em execução no servidor. Infelizmente, essa funcionalidade não pôde ser implementada, então o sinal SIGINT apenas interrompe o processo do cliente. O outro sinal é um que envia EOF (End of File ou ^D), que foi implementado, e faz com que a execução do cliente pare e este seja desconectado do servidor;

7. Cliente

O programa cliente se conecta com o servidor por meio de um socket. Ele possui diversas funções, como as de manter logs de envios e recebimentos de mensagens, mudar portas e hosts para conexão, compressão e descompressão de dados, além de tratamento de erros.

No cliente, temos uma divisão de tarefas em três threads. Uma é responsável pelo envio de mensagens, outra pelo recebimento de mensagens, e outra pelo controle dessas duas. Houve uso de um mutex para implementar exclusão mútua na espera pela resposta do servidor, antes de requisitar outro input do usuário.

8. Servidor

O programa servidor aceita as conexões dos clientes via socket. Ele possui diversas funções, como as de mudar porta para conexão, compressão e descompressão de dados, além de tratamento de erros.

No servidor, temos uma divisão de tarefas em (número de clientes + 1) threads. As threads dos clientes são responsáveis pelo envio de mensagens e recebimento de mensagens, além da compressão de dados. A outra thread serve para controlar as outras. Houve uso de um mutex para implementar exclusão mútua na adição e remoção de clientes em uma estrutura de fila usada na implementação, para evitar possíveis race conditions.

9. Problemas Encontrados

Durante os poucos dias em que foi feito esse desenvolvimento de projeto, foram encontrados alguns problemas, muitos dos quais foram superados. Os que não apresentaram soluções, seja graças à nossa falta de conhecimento, experiência ou tempo, foram tomados como limitações, e são discutidos no tópico 11 deste trabalho.

Um dos que foram resolvidos, no entanto, foi o de apenas conseguir ler a primeira ou última linha do resultado enviado pelo terminal para o servidor. Este problema se deu porque estávamos usando a função fgets() para ler o resultado e passá-lo ao buffer. No entanto, foi muito mais eficiente e funcional utilizar o fread(), que lê todo o resultado e o transfere para o buffer, até atingir seu limite máximo.

Outro problema recorrente foi que a primeira mensagem recebida pelo cliente sempre apresentava algum lixo de memória no final, chegando até mesmo a dar erro no programa em algumas ocasiões. Não sabemos ao certo como esse problema foi

resolvido, e apenas constatamos que ele desapareceu após transformar os comandos responsáveis pela leitura do resultado em uma função à parte.

10. Possíveis Melhorias

Como o prazo dado para implementação do projeto foi curto, ele está mais próximo de um protótipo do que de uma versão final. Há diversas melhorias que podem ser adicionadas a ele, entre as quais podemos incluir: uso de múltiplos buffers pequenos ao invés de um grande, para melhor alocação de memória;

11. Limitações

Uma limitação encontrada foi a de não poder utilizar comandos que exigem interação com o usuário, como por exemplo, qualquer comando precedido de sudo, exigindo a senha do root. Ainda mais a implementação…

Resources

Tomar, Nikhil. (2021) “Chatroom in C", https://github.com/nikhilroxtomar/Chatroom-in-C, July.