

Compressed Network Communication

Guilherme L. da Silva Alves, Lucas R. Alvino, Natália R. de Almada

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

`liriooberto@gmail.com, lucasbass648@gmail.com, almada.compsci@gmail.com`

Abstract. *The main objective of this article is to record the development of the final project of the operating systems discipline, the “Compressed Network Communication” project, which can be developed in one of the following three languages: C, C++ or Rust. In this case, they would have to develop two programs, which would be two processes, a client and a server, in which the project would communicate from network sockets through the TCP/IP protocol, in addition, there was the use of the zlib library for compression as messages between these processes.*

Resumo. *Este artigo tem como objetivo principal registrar o desenvolvimento do projeto final da disciplina sistemas operacionais, o projeto “Compressed Network Communication”, o qual poderia ser desenvolvido em uma das três linguagens a seguir: C, C++ ou Rust. Nesse projeto deveríamos desenvolver dois programas, que seriam dois processos um cliente e um servidor, no qual deveriam se comunicar através de sockets de rede através do protocolo TCP/IP, além disso deveria utilizar a biblioteca zlib para a compactação, ao adicionar por meio da linha de comando a opção --compress ao cliente e ao servidor, ativará a compactação de todo o tráfego em ambas as direções.*

1. Visão Geral

O projeto a ser desenvolvido tinha como objetivo a implementação da CNC (Compressed Network Communication), a comunicação entre os processos cliente e servidor seria por meio dos sockets TCP/IP, e a comunicação entre o servidor e o shell seria por meio do pipe.

2. Organização do projeto

Em relação a divisão de funções para a implementação do projeto decidimos da seguinte forma: Lucas Ribeiro e Guilherme ficaram responsáveis pela codificação do cliente e servidor, utilizamos a linguagem C e a IDE Visual Studio Code para escrever os códigos, Guilherme ficou responsável pelo slide para a apresentação, o slide foi construído usando o software Canva, Lucas Ribeiro e Natália Almada responsáveis pelo relatório, utilizamos o Google Docs .

3. Cliente

O programa cliente pode se conectar com servidor através do socket, como não conseguimos implementar as funções necessárias contidas na descrição do projeto, o nosso programa cliente é capaz manter uma comunicação semelhante a um chat, além de registrar todas as mensagens em um arquivo txt disponível no final da conversa com o servidor.

O arquivo .txt é gerado no início da conversa e as mensagens que o cliente enviar para o servidor quanto às mensagens que o cliente receber serão salvas no arquivo, esse “bate papo” é mantido enquanto o cliente ou o servidor não digitar “Até” na conversa. Além disso, tanto o cliente quanto o servidor possuem funções que manipulam e formatam a hora, com isso conseguimos mostrar em tempo real o horário em que as mensagens foram enviadas.

4. Servidor

O programa servidor é capaz de receber conexões do cliente, o nosso servidor pode aceitar no máximo 10 conexões inicialmente, podemos aumentar e diminuir o número de clientes após uma análise, semelhante ao cliente ele pode enviar mensagens para um cliente específico o primeiro que se conectar a ele, os clientes subsequentes podem se conectar entretanto não podem enviar mensagem para o servidor.

O servidor de semelhante forma ao cliente é capaz de registrar em um arquivo .txt as mensagens enviadas quanto as mensagens recebidas do cliente.

5. Socket

Socket (ou soquete de rede) é uma interface de software (API) cuja função é intermediar o envio e recebimento de mensagens de um processo, sendo assim, é a interface entre a camada de aplicação e a de transporte dentro de um hospedeiro, funcionando como uma porta. Um adendo é que eles podem ou não estar na mesma máquina. Ex: requisitar páginas web.

6. Fork

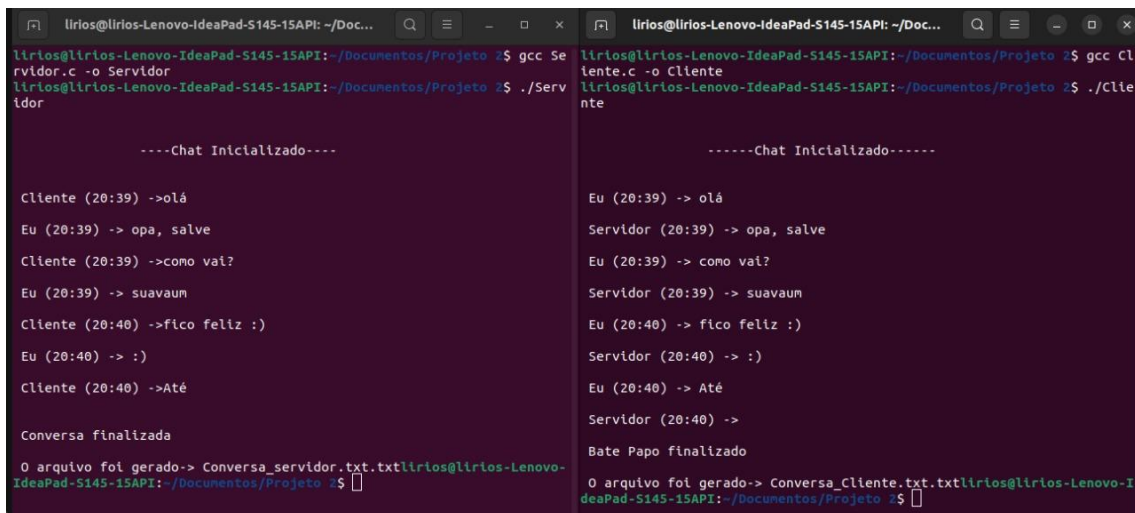
É uma função que vai duplicar o processo ocorrente dentro do sistema operacional (SO). O processo que a princípio chama a função fork é o que se chama de “processo-pai”, portanto, o novo processo criado pela função denominada “processo-filho”. O processo é duplicado e sua completude no SO, ou seja, pilhas, memória dinâmica, código, dados. A função é chamada apenas uma vez no pai e retorna duas vezes(uma no pai e outra no filho). O filho herda assim as informações contidas no processo-pai.

7. Pipe

Pipe (Cano) se refere a um par de arquivos que é criado num "processo pai-filho". O pipe conecta os processos resultantes quando o "processo pai" realiza a cópia de um repositório(fork). O próprio pipe não existe de fato em nenhuma pasta, então é dito como anônimo. Ele geralmente conecta apenas dois processos, contudo, é possível que qualquer quantidade de "processos-filho" possam se conectar um ao outro e ao seu "pai" por um mesmo pipe.

8. Resultados

Nessa imagem conseguimos mostrar o chat em tempo real, através de dois terminais, no terminal do lado direito temos o código do cliente compilado e já com a comunicação com o servidor, o terminal do lado esquerdo temos o servidor compilado e se comunicando com cliente.



```
lirios@lirios-Lenovo-IdeaPad-S145-15API: ~/Documentos/Projeto 2$ gcc Se
rvidor.c -o Servidor
lirios@lirios-Lenovo-IdeaPad-S145-15API: ~/Documentos/Projeto 2$ ./Serv
idor

----Chat Inicializado----

Cliente (20:39) ->olá
Eu (20:39) -> opa, salve
Cliente (20:39) ->como vai?
Eu (20:39) -> suavaum
Cliente (20:40) ->fico feliz :)
Eu (20:40) -> :)
Cliente (20:40) ->Até

Conversa finalizada

O arquivo foi gerado-> Conversa_servidor.txt.txtlirios@lirios-Lenovo-
IdeaPad-S145-15API: ~/Documentos/Projeto 2$

lirios@lirios-Lenovo-IdeaPad-S145-15API: ~/Documentos/Projeto 2$ gcc Cl
iente.c -o Cliente
lirios@lirios-Lenovo-IdeaPad-S145-15API: ~/Documentos/Projeto 2$ ./Clie
nte

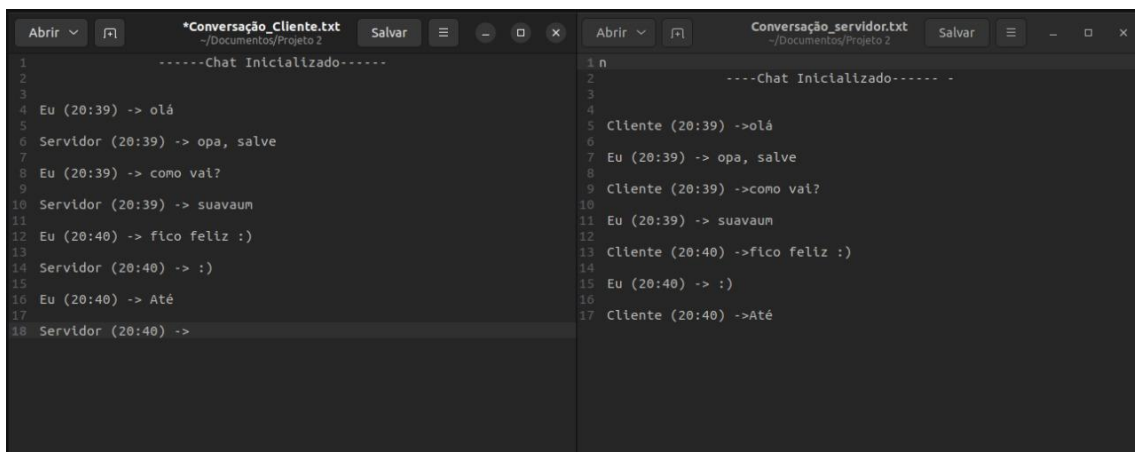
-----Chat Inicializado-----

Eu (20:39) -> olá
Servidor (20:39) -> opa, salve
Eu (20:39) -> como vai?
Servidor (20:39) -> suavaum
Eu (20:40) -> fico feliz :)
Servidor (20:40) -> :)
Eu (20:40) -> Até
Servidor (20:40) ->

Bate Papo finalizado

O arquivo foi gerado-> Conversa_Cliente.txt.txtlirios@lirios-Lenovo-I
deaPad-S145-15API: ~/Documentos/Projeto 2$
```

Já nesta imagem podemos observar um arquivos .txt nesses arquivos, estão registradas as mensagens que acontecem durante o bate papo no chat.



```
*Conversa_Cliente.txt
1 -----Chat Inicializado-----
2
3
4 Eu (20:39) -> olá
5
6 Servidor (20:39) -> opa, salve
7
8 Eu (20:39) -> como vai?
9
10 Servidor (20:39) -> suavaum
11
12 Eu (20:40) -> fico feliz :)
13
14 Servidor (20:40) -> :)
15
16 Eu (20:40) -> Até
17
18 Servidor (20:40) ->

Conversa_servidor.txt
1 n
2 -----Chat Inicializado-----
3
4
5 Cliente (20:39) ->olá
6
7 Eu (20:39) -> opa, salve
8
9 Cliente (20:39) ->como vai?
10
11 Eu (20:39) -> suavaum
12
13 Cliente (20:40) ->fico feliz :)
14
15 Eu (20:40) -> :)
16
17 Cliente (20:40) ->Até
```

9. Dificuldades encontradas

Inicialmente as dificuldades que encontramos foi quando tentamos implementar a biblioteca zlib para a compressão dos dados, também em relação aos comandos necessários no projeto, não conseguimos implementar em nossos códigos.

Outra dificuldade essa em relação aos nossos códigos em si, foi que quando começamos a utilizá-los eles estavam com alguns erros em algumas funções, os códigos estavam compilando entretanto apresentavam erros, entretanto conseguimos solucionar os erros.

10. Considerações finais

Após apresentarmos todos os componentes do nosso trabalho bem como os códigos fontes e os resultados, concluímos que apesar das inúmeras dificuldades que encontramos no desenvolvimento do projeto não conseguindo implementar muitas funcionalidades necessárias, o fato de termos procurando e tentando desenvolver o projeto e a experiência e o aprendizado que adquirimos foram e serão importantes para o crescimento do nosso conhecimento, tanto da disciplina de sistemas operacionais I quanto a disciplina de redes de computadores I.

Referência

Kennedy Tedesco. Uma introdução a TCP, UDP e Sockets. treinaweb. 2019. Disponível em: <https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets>. Acesso em: 15 de julho de 2022

SÁNCHEZ, Rodrigo. Cómo crear un chat en C utilizando Sockets TUTORIAL. Youtube, 17 de janeiro de 2021. Disponível em: <https://www.youtube.com/watch?v=kL1RVAQWA7o>. Acesso em: 15 de julho de 2022.

BIRODKAR, Vighnesh . C Socket Tutorial – Echo Server. A Simple Programmer 's Blog. 14 de março de 2013. Disponível em: <https://vcansimplify.wordpress.com/2013/03/14/c-socket-tutorial-echo-server/>. Acesso em: 18 de julho de 2022

UNIRIO / CCET - Ensino e Pesquisa - Produzir e disseminar conhecimento. Guia Linux, [https://guialinux.uniriotec.br/arquivo/#:~:text=de%20baixo%20n%C3%ADvel\).-,Descritor%20de%20arquivo,%2Fusr%2Finclude%2Funistd](https://guialinux.uniriotec.br/arquivo/#:~:text=de%20baixo%20n%C3%ADvel).-,Descritor%20de%20arquivo,%2Fusr%2Finclude%2Funistd).

KUROSE, J. F., ROSS, K. Redes de Computadores e a Internet: uma abordagem top-down. Edição: 6ª, Editora Pearson, 2014.

SPADARI, Ana. O que faz um fork?. CCM. 12 de janeiro de 2021. Disponível em: <https://br.ccm.net/faq/10841-o-que-faz-um-fork>. Acesso em: 20 de julho de 2022

ORACLE.Programming Interfaces Guide. 2010. Pipes Between Processes. Disponível em: <https://docs.oracle.com/cd/E19683-01/806-4125/6jd7pe6bo/index.html>. Acesso em: 24 de julho de 2022