UNIVERSIDADE VEIGA DE ALMEIDA (UVA)

COORDENAÇÃO DO CURSO DE ENGENHARIA DA COMPUTAÇÃO – CAMPUS BARRA DA TIJUCA

RENAN CARVALHO COELHO

BIBLIOTECA EM LINGUAGEM C PARA A INFRAESTRUTURAÇÃO DE REDES

RIO DE JANEIRO, RJ

2024

RENAN CARVALHO COELHO

BIBLIOTECA EM LINGUAGEM C PARA A INFRAESTRUTURAÇÃO DE REDES

Trabalho de conclusão de curso, apresentado como requisito parcial para obtenção do título de Bacharel em Engenheira da Computação do curso de graduação em Engenharia da Computação da Universidade Veiga de Almeida.

ORIENTADOR: Prof. Miguel Ângelo Zaccur de Figueiredo, DSc.

RIO DE JANEIRO, RJ
2024

Coelho, Renan Carvalho

Biblioteca em linguagem C para a Infraestruturação de Redes / Renan Carvalho Coelho. - Rio de Janeiro, 2024.

86 p. : il. p&b.

Trabalho de conclusão de curso (graduação) – Universidade Veiga de Almeida, Curso de Engenharia da Computação, Rio de Janeiro, 2024.

Orientador: Miguel Ângelo Zaccur de Figueiredo

1. Roteamento IP. 2. C. 3. Redes de Computadores. I. Ângelo, Miguel. II. Universidade Veiga de Almeida. III. BIBLIOTECA EM LINGUAGEM C PARA A INFRAESTRUTURAÇÃO DE REDES.

UNIVERSIDADE VEIGA DE ALMEIDA - UVA

BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

RENAN CARVALHO COELHO

BIBLIOTECA EM LINGUAGEM C PARA A INFRAESTRUTURAÇÃO DE REDES

Monografia apresentada como requisito parcial à conclusão do curso em Bacharel em Engenharia da Computação.

APROVADA EM: 05/12/2024

CONCEITO: 10 (dez)

BANCA EXAMINADORA:

PROF. DSc MIGUEL ÂNGELO ZACCUR DE FIGUEIREDO ORIENTADOR

PROF. MSc CAMILLA LOBO PAULINO

PROF. DSc THIAGO ALBERTO RAMOS GABRIEL

RENAN CARVALHO COELHO

Coordenação de Engenharia da Computação

Rio de Janeiro, 05 de dezembro de 2024

AGRADECIMENTOS

Este trabalho é dedicado a você, você sabe quem você é,
e, sem você, este trabalho não existiria.

BIBLIOTECA EM LINGUAGEM C PARA A INFRAESTRUTURAÇÃO DE REDES

Renan Carvalho Coelho [1]

RESUMO

O presente trabalho descreve o desenvolvimento de uma biblioteca em linguagem C voltada ao auxílio de tarefas fundamentais na infraestrutura de redes de computadores. Com funções dedicadas à manipulação de endereços IPv4 e IPv6, tratamento de endereços MAC, cálculo de máscaras de sub-rede e CIDR, a biblioteca destina-se tanto a fins didáticos quanto ao suporte de desenvolvedores que atuem no desenvolvimento de aplicações para o setor de redes. Concebida com um viés educacional, a biblioteca busca oferecer uma ferramenta prática que complemente o ensino teórico de redes, alinhando-se às necessidades de profissionais em formação e contribuindo para uma compreensão mais sólida dos elementos de infraestrutura de redes. As redes de computadores surgiram com o objetivo de interconectar máquinas, evoluindo de pequenas redes locais até a criação da internet global. Segundo Tannenbaum e Wetherall (2011), "uma rede de computadores é um conjunto de computadores autônomos interconectados por uma única tecnologia", definição que ilustra o núcleo das redes e sua expansão histórica. Com a necessidade de atender ao crescimento exponencial da comunicação digital, protocolos como IPv4 foram estabelecidos, seguidos mais recentemente pelo IPv6, para lidar com a limitação de endereços. Além disso, conceitos como sub-redes e o modelo CIDR foram introduzidos para organizar e otimizar a utilização dos espaços de endereço. Esse cenário de evolução impulsiona a criação de ferramentas práticas que acompanhem e auxiliem as demandas de infraestrutura e compreensão das redes modernas. A biblioteca desenvolvida neste trabalho representa, portanto, um recurso valioso que se integra ao contexto educacional e prático das redes de computadores, oferecendo suporte tanto para o aprendizado quanto para a criação de soluções voltadas à infraestruturação de redes. Com funções que permitem a manipulação prática de dados de rede e abstração de operações essenciais relacionadas as redes de computadores, a biblioteca é capaz de enriquecer o conhecimento técnico já existente no campo de estudo correspondente e de otimizar o trabalho de desenvolvedores e técnicos da área. Em conclusão, este projeto reflete o esforço contínuo pela formação de profissionais qualificados, pela eficiência na gestão de redes e pela promoção de uma infraestrutura de redes global segura e adaptável, alinhada com a complexidade do ambiente digital atual.

Palavras-chave: Infraestrutura de redes; Redes de computadores; IPv4; IPv6; Máscara de sub-rede; CIDR; MAC; Linguagem C.

[1] Graduando em Engenharia da Computação, Universidade Veiga de Almeida, Campus Barra da Tijuca, Rio de Janeiro.

ABSTRACT

This work describes the development of a C language library aimed at supporting fundamental tasks in Computer Networking. Including functions dedicated to IPv4 and IPv6 address manipulation, MAC address handling, subnetting and CIDR calculation, the library is intended both for educational purposes and to assist developers creating applications for the sector. Designed with an educational focus, the library seeks to offer a practical tool that complements theoretical computer networks teaching, aligned with the needs of aspiring professionals and contributing to a more comprehensive understanding of networks infrastructuring elements. Computer networks emerged with the goal of interconnecting machines, evolving from small local networks to the creation of the global internet. According to Tannenbaum and Wetherall (2011), "a computer network is a set of autonomous computers interconnected by a single technology", a definition that illustrates the core fundament of networks and their historical expansion. With the need to support the exponential growth of digital communication, protocols such as IPv4 were established, followed more recently by IPv6, which emerged to address the limitation of available addresses. Additionally, concepts such as subnetting and the CIDR model were introduced to organize and optimize address space usage. This evolving scenario pushes the creation of practical tools that meet and assist the demands that emerge with modern network infrastructures and their understanding. The library developed in this work thus represents a valuable resource that integrates itself into the educational and practical context of computer networks, offering support both for learning and for the creation of network infrastructure solutions. Alongside functions that enable the practical manipulation of network data and the abstraction of essential operations related to computer networks, the library can enrich already existing technical knowledge in the field and optimize the work of developers and technicians. In conclusion, this project reflects the ongoing effort to train qualified professionals, enhance computer networks management efficiency, and promote a secure and adaptable networks global infrastructure, aligned with the complexities of today's digital environment.

Keywords: Computer network; Subnetwork; Subnet Mask; Routing; IPv4; IPv6; CIDR; MAC Adress; C Language.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

As redes de computadores desempenham um papel fundamental na sociedade contemporânea, facilitando a comunicação e a troca de informações entre dispositivos em uma escala global. Com o crescimento exponencial da Internet e a interconexão de dispositivos através da Internet das Coisas (IoT), a complexidade das redes aumentou significativamente. Isso gerou uma demanda crescente por ferramentas que ajudem tanto profissionais quanto estudantes a entender e trabalhar com esses sistemas de forma eficaz (BHATTI *et al*., 2022).

A linguagem C, sendo uma das linguagens de programação mais utilizadas para o desenvolvimento de software de sistemas, é particularmente adequada para a criação de bibliotecas que manipulam dados de rede. Sua eficiência em termos de desempenho e controle sobre os recursos do sistema torna-a uma escolha ideal para aplicações que exigem manipulação direta de memória e operações de baixo nível, características comuns em projetos relacionados a redes (ZIKRIA *et al*., 2022).

1.1. Contexto e Justificativa

A biblioteca network_utils foi desenvolvida com o intuito de simplificar operações comuns em redes, como conversões entre formatos de endereços IP (IPv4 e IPv6), cálculos de sub-redes e manipulação de dados binários. Ao fornecer funções que encapsulam essas operações complexas, a biblioteca não apenas facilita o trabalho dos desenvolvedores, mas também serve como uma ferramenta educacional valiosa para estudantes que estão aprendendo sobre redes e programação. Como destacado na literatura, a gestão eficaz de endereços IP e operações relacionadas é um componente essencial para o funcionamento de redes modernas e escaláveis (ROONEY; DOOLEY, 2021).

Além disso, a implementação da biblioteca foi realizada tanto em um ambiente WSL Ubuntu, quanto pelo terminal MSYS2 com Mingw64 integrado, utilizando o compilador GCC, o que garante compatibilidade com sistemas operacionais Windows e Linux. A biblioteca foi testada rigorosamente através de um arquivo separado, assegurando que suas funções operem corretamente em diferentes cenários.

A justificativa para o desenvolvimento da network_utils se baseia na necessidade de um recurso acessível e eficiente que ajude a desmistificar conceitos complexos relacionados às redes. Ao tornar essas operações mais acessíveis, espera-se que mais estudantes e profissionais possam se envolver com o desenvolvimento em redes, contribuindo assim para um melhor entendimento e aplicação das tecnologias atuais.

1.2. Objetivos

1.2.1. Objetivo Geral

Desenvolver uma biblioteca em C que forneça funções para manipulação e conversão de endereços IP, além de cálculos relacionados a redes.

1.2.2. Objetivos Específicos

1.2.2.1. *Implementar funções para conversão entre formatos decimal, binário e hexadecimal*

A biblioteca deve incluir funções que permitam aos usuários converterem números entre diferentes representações, facilitando a compreensão dos dados subjacentes às operações em redes.

1.2.2.2. *Criar funções para calcular endereços de rede, broadcast e máscaras CIDR*

Essas funções são essenciais para o gerenciamento eficiente das sub-redes em uma rede IP, permitindo que os usuários determinem rapidamente as características da rede em que estão trabalhando.

1.2.2.3. *Proporcionar uma ferramenta educacional que ajude estudantes a compreender melhor os conceitos fundamentais das redes de computadores*

A biblioteca deve ser utilizada como um recurso didático em cursos relacionados à tecnologia da informação, permitindo que os alunos experimentem com código real enquanto aprendem sobre conceitos teóricos.

Esses tópicos oferecem uma visão mais detalhada do contexto e justificativa do desenvolvimento da biblioteca network_utils, bem como dos objetivos gerais e específicos do trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

As redes de computadores são sistemas que permitem a interconexão de dispositivos para a troca de informações. Esses sistemas são fundamentais para a comunicação moderna e são utilizados em diversas aplicações, desde redes locais em residências até grandes infraestruturas que suportam a Internet. A seguir, abordaremos os conceitos fundamentais relacionados às redes de computadores, incluindo suas estruturas, protocolos, endereçamento IP, máscaras de sub-rede e CIDR, além de uma visão histórica do desenvolvimento das redes e das principais empresas do setor do desenvolvimento. A compreensão de conceitos fundamentais, como os modelos de rede e suas arquiteturas, é essencial para acompanhar os avanços em segurança, integração da Internet das Coisas (IoT) e computação em borda, que moldam as redes modernas (GUDHKA, 2024).

2.1. Conceitos de Redes de Computadores

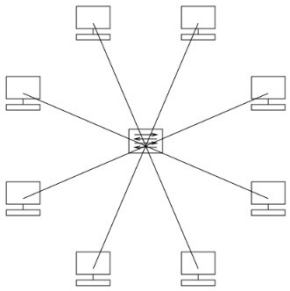
2.1.1. Estruturas de Rede (Topologias)

A topologia de uma rede refere-se à disposição física ou lógica dos dispositivos conectados. As principais topologias incluem:

2.1.1.1. *Topologia em Estrela*

Todos os dispositivos estão conectados a um ponto central, geralmente um switch ou hub. Essa configuração é fácil de gerenciar e isolar problemas, mas depende da funcionalidade do dispositivo central. Essa topologia é amplamente utilizada em ambientes corporativos devido à sua facilidade de manutenção.

Figura 1 – ***Čeština***



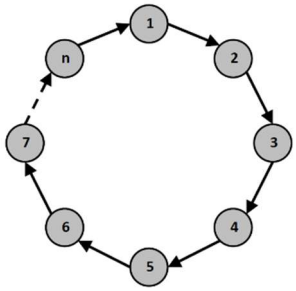
Fonte: (QEEF, 2015).

2.1.1.2. *Topologia em Anel*

Cada dispositivo está conectado a vários outros dispositivos. Isso proporciona redundância e maior confiabilidade, já que há múltiplos caminhos para os dados. No

entanto, essa configuração pode ser complexa e cara devido ao número elevado de conexões.

Figura 2 – ***Ring network scheme***

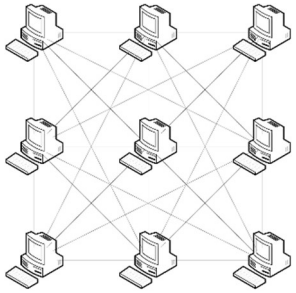


Fonte: (SIMEONOVSKI, 2017).

2.1.1.3. *Topologia em Malha*

Os dispositivos estão conectados em um formato circular. Cada dispositivo tem exatamente dois vizinhos para a comunicação. A transmissão de dados ocorre em uma direção específica. Essa topologia pode ser mais difícil de gerenciar, pois uma falha em um único dispositivo pode afetar toda a rede.

Figura 3 – ***Diagram Of A True Mesh Topology Network***

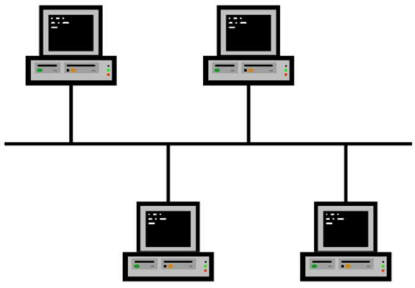


Fonte: (KOMAN90, 2009).

2.1.1.4. *Topologia em Barramento*

Todos os dispositivos compartilham um único cabo central. Embora seja uma configuração simples e econômica, ela é suscetível a colisões e falhas no cabo central podem derrubar toda a rede.

Figura 4 – ***Netzwerktopologie Bus***



Fonte: (HEAD, 2023).

Essas topologias podem ser combinadas para formar redes híbridas que aproveitam as vantagens de diferentes configurações (TANENBAUM; WETHERALL, 2011).

2.1.2. Protocolos (TCP/IP)

Os protocolos são conjuntos de regras que definem como os dados são transmitidos e recebidos em uma rede. O conjunto de protocolos mais utilizado é o TCP/IP (*Transmission Control Protocol/Internet Protocol*), que é a base da Internet. O TCP/IP é dividido em quatro camadas principais:

2.1.2.1. *Camada de Aplicação*

Onde os aplicativos interagem com a rede (ex: HTTP, FTP).

2.1.2.2. *Camada de Transporte*

Responsável pela entrega confiável dos dados (ex: TCP, UDP).

2.1.2.3. *Camada de Internet*

Trata do endereçamento e roteamento dos pacotes (ex: IP).

2.1.2.4. *Camada de Acesso à Rede*

Gerencia as interfaces físicas e as tecnologias utilizadas para transmissão.

O protocolo IP é responsável por endereçar pacotes de dados entre dispositivos na rede, enquanto o TCP garante que esses pacotes sejam entregues corretamente e na ordem certa (TANENBAUM; WETHERALL, 2011).

2.1.3. Endereçamento IP (IPv4 e IPv6)

O endereçamento IP é fundamental para a comunicação em redes. Cada dispositivo precisa ter um endereço IP único para se comunicar com outros dispositivos na rede (TANENBAUM; WETHERALL, 2011).

2.1.3.1. *IPv4*

O protocolo IPv4 utiliza endereços de 32 bits, permitindo aproximadamente 4 bilhões de endereços únicos (2^32). Os endereços são geralmente representados em notação decimal pontuada (ex: 192.168.0.1). No entanto, com o crescimento da Internet, o espaço de endereços IPv4 se tornou insuficiente.

2.1.3.2. *IPv6*

Para resolver a limitação do IPv4, foi desenvolvido o IPv6, que utiliza endereços de 128 bits, permitindo um número praticamente ilimitado de endereços (2^128). Os

endereços IPv6 são representados em notação hexadecimal e separados por dois pontos (ex: 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

2.1.4. Máscaras de Sub-Rede e CIDR

As máscaras de sub-rede são utilizadas para dividir uma rede maior em sub-redes menores, permitindo uma melhor organização e gerenciamento do tráfego na rede. A máscara determina quais partes do endereço IP representam a rede e quais representam o host (TANENBAUM; WETHERALL, 2011).

2.1.4.1. *Máscara Padrão*

Uma máscara padrão para IPv4 pode ser 255.255.255.0, que indica que os primeiros três octetos representam a rede e o último octeto representa os hosts.

2.1.4.2. *CIDR (Classless Inter-Domain Routing)*

O CIDR permite uma alocação mais flexível dos endereços IP através da notação "slash" (ex: /24), onde o número indica quantos bits estão sendo usados para identificar a parte da rede do endereço IP. Isso permite uma utilização mais eficiente do espaço de endereços disponíveis.

2.1.5. História das Redes de Computadores

A história das redes de computadores remonta à década de 1960, quando o Departamento de Defesa dos Estados Unidos desenvolveu a ARPANET, uma das primeiras redes a utilizar pacotes para transmitir dados entre computadores. Essa inovação levou ao desenvolvimento do protocolo TCP/IP nos anos 1970, permitindo que diferentes redes fossem interconectadas e estabelecendo a base para a Internet moderna. Como observado, “o desenvolvimento da ARPANET e sua transição para o uso do TCP/IP em 1983 não apenas revolucionou as redes, mas criou um padrão universal de comunicação digital” (CROCKER, 2022).

Na década seguinte, surgiram as primeiras implementações comerciais das redes locais (LANs), utilizando tecnologias como Ethernet e Token Ring. A popularização da Internet na década de 1990 impulsionou ainda mais o desenvolvimento das redes globais.

Empresas como Cisco, Juniper e Huawei emergiram como líderes no setor:

2.1.5.1. *Cisco Systems*

Fundada em 1984, Cisco se tornou uma das principais fornecedoras de equipamentos e soluções para redes corporativas e operadoras. A empresa é conhecida por seus roteadores e switches que formam a espinha dorsal da Internet moderna.

2.1.5.2. *Juniper Networks*

Fundada em 1996, Juniper é conhecida por suas soluções avançadas em roteamento e segurança para provedores de serviços e grandes empresas. A empresa se destaca pela inovação em hardware e software para gerenciamento eficiente das redes.

2.1.5.3. *Huawei Technologies*

Fundada na China em 1987, Huawei cresceu rapidamente para se tornar um dos maiores fornecedores globais de equipamentos telecomunicações e tecnologia da informação. A empresa investe fortemente em pesquisa e desenvolvimento, oferecendo soluções abrangentes para redes móveis e fixas.

Essas empresas desempenham papéis cruciais no desenvolvimento contínuo das tecnologias de rede, contribuindo para inovações que tornam as comunicações mais rápidas, seguras e eficientes (HOWARD, 2017).

2.2. Linguagem C e Ferramentas para o Desenvolvimento

A linguagem C é uma das linguagens de programação mais influentes e amplamente utilizadas no desenvolvimento de software. Criada por Dennis Ritchie no início da década de 1970, a linguagem foi projetada para ser simples, eficiente e portátil, permitindo que desenvolvedores escrevessem programas que pudessem ser executados em diferentes plataformas. A linguagem C é frequentemente utilizada em sistemas operacionais, compiladores, e aplicações que exigem alto desempenho e controle sobre os recursos do sistema, "A linguagem C, desenvolvida por Dennis Ritchie na década de 1970, foi fundamental para a criação do sistema UNIX, um dos pilares da tecnologia moderna, e permanece amplamente utilizada em sistemas que demandam alta eficiência e controle" (Computer History Museum, 2024).

2.2.1. História da Linguagem C

A história da linguagem C remonta a 1972, quando foi desenvolvida na Bell Labs como uma evolução da linguagem B. O principal objetivo era criar uma linguagem que pudesse ser utilizada para implementar o sistema operacional UNIX, que também estava em desenvolvimento na época. Desde então, a linguagem C evoluiu e se tornou um padrão na indústria de software (RITCHIE, 1993).

Em 1989, a *American National Standards Institute* (ANSI) aprovou o padrão ANSI C, que formalizou a linguagem e garantiu sua portabilidade entre diferentes sistemas. Em 1999, uma nova versão conhecida como C99 foi lançada, introduzindo várias melhorias, como suporte a variáveis de comprimento variável e novos tipos de

dados. Mais recentemente, o padrão C11 trouxe ainda mais recursos, incluindo suporte a *multithreading* e melhorias na segurança.

2.2.2. Vantagens da Linguagem C

2.2.2.1. *Desempenho*

A linguagem C é conhecida por sua eficiência em termos de execução e uso de memória. Isso a torna ideal para aplicações que exigem alto desempenho.

2.2.2.2. *Portabilidade*

Programas escritos em C podem ser facilmente transferidos entre diferentes plataformas com poucas ou nenhuma modificação no código-fonte.

2.2.2.3. *Controle sobre Recursos*

A linguagem permite acesso direto à memória através do uso de ponteiros, oferecendo aos desenvolvedores um controle granular sobre os recursos do sistema.

2.2.2.4. *Ampla Utilização*

Devido à sua popularidade, existe uma vasta quantidade de bibliotecas e frameworks disponíveis para C, facilitando o desenvolvimento.

2.2.3. Desvantagens da Linguagem C

2.2.3.1. *Complexidade*

A linguagem C é conhecida por sua eficiência em termos de execução e uso de memória. Isso a torna ideal para aplicações que exigem alto desempenho.

2.2.3.2. *Falta de Recursos Modernos*

Programas escritos em C podem ser facilmente transferidos entre diferentes plataformas com poucas ou nenhuma modificação no código-fonte.

2.2.3.3. *Dificuldade em Aprender*

Para iniciantes, a curva de aprendizado pode ser íngreme devido à necessidade de entender conceitos como ponteiros e alocação dinâmica de memória.

Para programar em C, diversas ferramentas estão disponíveis que facilitam o desenvolvimento e a compilação do código. Entre as mais relevantes estão o GCC (*GNU Compiler Collection*), MinGW64, WSL (*Windows Subsystem for Linux*) e MSYS2.

2.2.4. GCC (*GNU Compiler Collection*)

O GCC é um dos compiladores mais populares e amplamente utilizados para a linguagem C. Desenvolvido como parte do projeto GNU, o GCC suporta várias linguagens além do C, incluindo C++, Fortran e Ada. Ele é conhecido por sua portabilidade e pela conformidade com os padrões ANSI/ISO.

O GCC é frequentemente utilizado em ambientes Linux e *Unix-like* devido à sua robustez e flexibilidade. Além disso, ele oferece uma ampla gama de opções de otimização que permitem aos desenvolvedores melhorarem o desempenho do código gerado (FREE SOFTWARE FOUNDATION, 2024).

2.2.5. MinGW64

MinGW64 (Minimalist GNU for Windows) é uma variante do GCC projetada para compilar aplicativos nativos do Windows. Ele fornece um ambiente leve para desenvolver software em C/C++ no Windows sem depender do ambiente completo do Windows API.

MinGW64 é particularmente útil para desenvolvedores que desejam criar aplicativos multiplataforma com um único conjunto de ferramentas. Ele permite que os desenvolvedores utilizem as bibliotecas GNU enquanto ainda aproveitam as funcionalidades específicas do Windows (ROMERO, 2024).

2.2.6. WSL (*Windows Subsystem for Linux*)

O WSL permite que usuários do Windows executem um ambiente Linux diretamente no Windows sem a necessidade de uma máquina virtual ou dual boot. Com o WSL, os desenvolvedores podem usar distribuições Linux populares como Ubuntu, Debian ou Fedora diretamente no Windows (WOJCIAKOWSKI et al; 2024).

Ao usar o GCC dentro do WSL, os desenvolvedores podem programar em C em um ambiente semelhante ao Linux, aproveitando as ferramentas e bibliotecas disponíveis nesse sistema operacional. Isso facilita o desenvolvimento de software que será executado em servidores Linux ou em ambientes onde o Linux é predominante.

2.2.7. MSYS2

O MSYS2 é um ambiente leve que fornece uma coleção de ferramentas GNU para Windows. Ele combina as funcionalidades do MinGW com um sistema de pacotes semelhante ao Arch Linux. O MSYS2 permite que os desenvolvedores instalem facilmente bibliotecas adicionais e ferramentas necessárias para o desenvolvimento em C (MSYS2, 2024).

Ao utilizar o GCC com MSYS2 pelo MinGW64, os desenvolvedores podem criar aplicativos nativos do Windows enquanto mantêm um fluxo de trabalho familiar aos usuários de Linux. O MSYS2 também suporta *scripts shell* e outras ferramentas comuns no desenvolvimento em ambientes Unix-like.

3. DESENVOLVIMENTO DA BIBLIOTECA

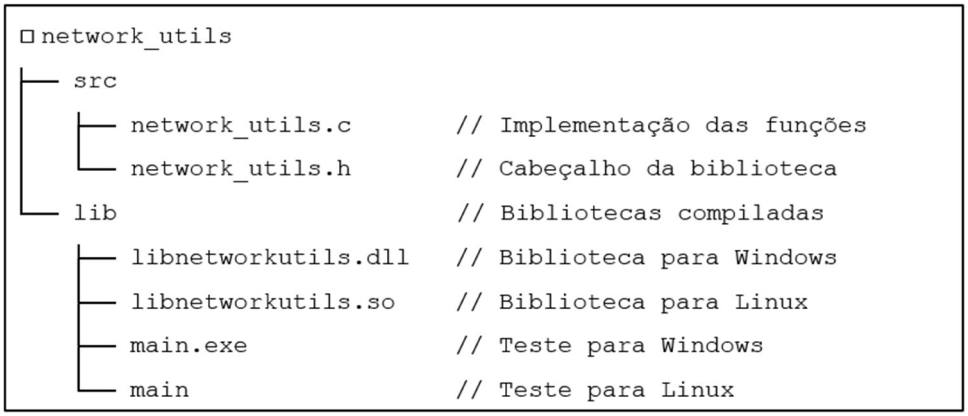
3.1. Estrutura e Organização do Projeto

A biblioteca desenvolvida, denominada *Network Utilities*, é uma implementação modularizada em C que visa fornecer uma série de funções utilitárias para manipulação e análise de endereços IP, CIDR, roteamento e diagnóstico de rede. A seguir, detalha-se a estrutura e a organização do projeto, enfatizando a modularidade e a integração das bibliotecas utilizadas.

3.1.1. Estrutura de Diretórios

A estrutura básica do projeto é organizada da seguinte forma:

Figura 5 – **Estrutura do Projeto**



Fonte: elaborado pelo autor (2024)[2]

3.1.2. Arquivos Principais

3.1.2.1. *network_utils.h*

Este arquivo contém as declarações das funções disponíveis na biblioteca, bem como definições de estruturas e macros. A modularização é evidenciada pela separação clara entre as declarações e suas implementações. As funções estão agrupadas por categorias, como conversão de endereços IP, operações CIDR, roteamento e diagnósticos de rede.

3.1.2.2. *network_utils.c*

Neste arquivo, são implementadas as funções declaradas no cabeçalho. A implementação é realizada com foco na clareza e eficiência, utilizando práticas recomendadas de programação em C. Cada função é responsável por uma tarefa específica, o que facilita a manutenção e a extensão da biblioteca.

---

[2] Figura desenvolvida pelo autor para esquematização dos diretórios do projeto.

3.1.3. Compilação da Biblioteca

Para garantir a portabilidade da biblioteca entre diferentes sistemas operacionais, foram utilizados dois comandos distintos para compilar a biblioteca (HOCK-CHUAN, 2018).

3.1.3.1. *No Windows*

Utilizou-se o GCC no WSL com o seguinte comando "`gcc -shared -o libnetworkutils.dll network_utils.c -fPIC -lm -lgmp -lws2_32`"[3]. Este comando gera uma biblioteca dinâmica (.dll) que pode ser utilizada em aplicações Windows.

3.1.3.2. *No Linux*

Utilizou-se o GCC do MinGW64 no MSYS2 com o comando "`gcc -shared -o libnetworkutils.so network_utils.c -fPIC -lm -lgmp`". Este comando gera uma biblioteca compartilhada (.so) que pode ser utilizada em aplicações Linux.

3.1.4. Dependências Externas[4]

A biblioteca faz uso das seguintes bibliotecas externas:

3.1.4.1. *libm*

Para operações matemáticas.

3.1.4.2. *libgmp*

Para manipulação de números inteiros grandes, especialmente nas funções que requerem cálculos precisos.

3.1.4.3. *libws2_32*

Específica para Windows, utilizada para funcionalidades relacionadas à rede.

3.1.5. Modularidade e Reutilização

A modularidade do código é um aspecto fundamental do projeto. Cada função é projetada para ser independente, permitindo que desenvolvedores integrem apenas as partes necessárias em seus projetos. Além disso, a organização em arquivos separados facilita a reutilização do código em outros contextos ou projetos futuros.

---

[3] Fonte "Courier", tamanho 10, foi utilizada na monografia para representações de linhas de código, visto ser uma fonte padrão de terminais para Windows, Linux, dentre outros sistemas operacionais de forma geral.

[4] Para execução do código em ambiente Windows, será necessária a inclusão das DLLs ao path do sistema ou sua inclusão no diretório onde se localiza o executável do programa que utilize também a biblioteca *Network Utilities*.

A estrutura e organização da biblioteca *Network Utilities* foi cuidadosamente planejada para proporcionar uma base sólida para futuras implementações e extensões. A separação clara entre declarações e implementações, aliada à utilização de práticas modernas de programação em C, garante que a biblioteca seja não apenas funcional, mas também facilmente compreensível e extensível por outros desenvolvedores.

3.2. IMPLEMENTAÇÃO DAS FUNÇÕES

As principais funções implementadas na biblioteca incluem:

Tabela 1 – **Decimal para Binário**

Função	`decimal_to_binary`
Descrição	Converte um número decimal para uma *string* binária.
Exemplo	`char* bin = decimal_to_binary(13);` `printf("Binário: %s\n", bin);`
Saída do Exemplo	`Binário: 1101`

Fonte: elaborado pelo autor (2024)[5]

Tabela 2 – **Binário para Decimal**

Função	`binary_to_decimal`
Descrição	Converte uma *string* binária para um número decimal.
Exemplo	`int dec = binary_to_decimal("1101");` `printf("Decimal: %d\n", dec);`

[5] A seguir serão apresentadas mais 42 tabelas, cada uma abordando uma função diferente da biblioteca.

Saída do Exemplo	`Decimal: 13`

Fonte: elaborado pelo autor (2024)

Tabela 3 – **Hexadecimal para Binário**

Função	`hexadecimal_to_binary`
Descrição	Converte uma *string* hexadecimal para uma *string* binária.
Exemplo	`char* bin = hexadecimal_to_binary("AF12");` `printf("Binário: %s\n", bin);`
Saída do Exemplo	`Binário: 1010111100010010`

Fonte: elaborado pelo autor (2024)

Tabela 4 – **Binário para Hexadecimal**

Função	`binary_to_hexadecimal`
Descrição	Converte uma *string* binária para uma *string* hexadecimal.

Exemplo	`char * hex =`
	`binary_to_hexadecimal("1010111100010010");`
	`printf("Hexadecimal: %s\n", hex);`

Saída do Exemplo	`Hexadecimal: AF12`

Fonte: elaborado pelo autor (2024)

Tabela 5 – **Decimal para Hexadecimal**

Função	`decimal_to_hexadecimal`
Descrição	Converte um número decimal para uma *string* hexadecimal.
Exemplo	`char* hex = decimal_to_hexadecimal(12245);` `printf("Hexadecimal: %s\n", hex);`
Saída do Exemplo	`Hexadecimal: 2FD5`

Fonte: elaborado pelo autor (2024)

Tabela 6 – **Hexadecimal para Decimal**

Função	`hexadecimal_to_decimal`
Descrição	Converte uma *string* hexadecimal para um número decimal.
Exemplo	`int dec = hexadecimal_to_decimal("2FD5");` `printf("Decimal: %d\n", dec);`
Saída do Exemplo	`Decimal: 12245`

Fonte: elaborado pelo autor (2024)

Tabela 7 – **IP de Notação Decimal para Notação Binária**

Função	`ip_to_binary`
Descrição	Converte um endereço IP para uma *string* binária.

Exemplo	`char* bin = ip_to_binary("192.168.3.45");` `printf("Binário: %s\n", bin);`
Saída do Exemplo	`Binário: 11000000101010000000001100101101`

Fonte: elaborado pelo autor (2024)

Tabela 8 – **IP de Notação Binária para Notação Decimal**

Função	`binary_to_ip`
Descrição	Converte uma *string* binária para um endereço IP.

Exemplo	`char* ip = binary_to_ip(`
	`"11000000101010000000001100101101");`
	`printf("IP: %s\n", ip);`

Saída do Exemplo	`Binário: 11000000101010000000001100101101`

Fonte: elaborado pelo autor (2024)

Tabela 9 – **IPv6 de Notação Hexadecimal para Notação Binária**

Função	`ipv6_to_binary`
Descrição	Converte um endereço IPv6 para uma *string* binária.

Exemplo	`char* bin =`
	`ipv6_to_binary("2001:db8::7e12:5559");`
	`printf("Binário: %s\n", bin);`

Saída do Exemplo	`Binário: 00100000000000010000110110111000000000`
	`000000000000000000000000000000000000000`
	`000000000000000001111110000100100101010101`
	`011001`

Fonte: elaborado pelo autor (2024)

Tabela 10 – **IPv6 de Notação Binária para Notação Hexadecimal**

Função	`binary_to_ipv6`
Descrição	Converte uma *string* binária para um endereço IPv6.
Exemplo	`char* ipv6 = binary_to_ipv6("00100000000000010000110110111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001111110000100100101010101011001");` `printf("IPv6: %s\n", ipv6);`
Saída do Exemplo	`IPv6: 2001:db8::7e12:5559`

Fonte: elaborado pelo autor (2024)

Tabela 11 – **Resumo de IPv6**

Função	`summarize_ipv6`
Descrição	Simplifica a notação de um endereço IPv6.
Exemplo	`char*summarized_ipv6 = summarize_ipv6("2001:0db8:85a3:0000:0000:8a2e:0370:7334");` `printf("IPv6 Resumido: %s\n", summarized_ipv6);`
Saída do Exemplo	`IPv6 Resumido: 2001:db8:85a3::8a2e:370:7334`

Fonte: elaborado pelo autor (2024)

Tabela 12 – **Notação CIDR para Máscara de Sub-Rede**

Função	`cidr_to_mask`

Descrição	Converte da notação CIDR para uma máscara de sub-rede.
Exemplo	`char* mask = cidr_to_mask(24);` `printf("Máscara: %s\n", mask);`
Saída do Exemplo	`Máscara: 255.255.255.0`

Fonte: elaborado pelo autor (2024)

Tabela 13 – **Máscara de Sub-Rede para Notação CIDR**

Função	`mask_to_cidr`
Descrição	Converte uma máscara de sub-rede para a notação CIDR.
Exemplo	`int cidr = mask_to_cidr("255.255.255.0");` `printf("CIDR: %d\n", cidr);`
Saída do Exemplo	`CIDR: 24`

Fonte: elaborado pelo autor (2024)

Tabela 14 – **Cálculo de Endereço de Rede**

Função	`calculate_network_address`
Descrição	Calcula o endereço de rede dado o endereço IP e a notação CIDR.

Exemplo	`char* network_address =`
	`calculate_network_address("192.168.1.100", 16);`
	`printf("Endereço de rede: %s\n",`
	`network_address);`

Saída do Exemplo	`Endereço de rede: 192.168.0.0`

Fonte: elaborado pelo autor (2024)

Tabela 15 – **Cálculo de Endereço de Broadcast**

Função	`calculate_broadcast_address`
Descrição	Calcula o endereço de *broadcast* dado o endereço IP e a notação CIDR.
Exemplo	`char* broadcast_address = calculate_broadcast_address("192.168.1.100",16); printf("Endereço de broadcast: %s\n", broadcast_address);`
Saída do Exemplo	`Endereço de broadcast: 192.168.255.255`

Fonte: elaborado pelo autor (2024)

Tabela 16 – **Cálculo do Número de Hosts**

Função	`calculate_number_of_hosts`
Descrição	Calcula o número de hosts disponíveis em uma rede IP dada a notação CIDR.
Exemplo	`int number_of_hosts = calculate_number_of_hosts(24); printf("Número de hosts disponíveis: %d\n", number_of_hosts);`
Saída do Exemplo	`Número de hosts disponíveis: 254`

Fonte: elaborado pelo autor (2024)

Tabela 17 – **Cálculo do Primeiro Host Disponível em uma Rede IP**

Função	`calculate_first_host`
Descrição	Calcula o primeiro host disponível em uma rede IP dado o endereço de rede.
Exemplo	`char* first_host = calculate_first_host("192.168.3.0"); printf("Primeiro host disponível: %s\n", first_host);`
Saída do Exemplo	`Primeiro host disponível: 192.168.3.1`
Segundo Exemplo	`char* first_host = calculate_first_host( calculate_network_address("192.168.3.54", 24)); printf("Primeiro host disponível: %s\n", first_host);`
Saída do Segundo Exemplo	`Primeiro host disponível: 192.168.3.1`

Fonte: elaborado pelo autor (2024)

Tabela 18 – **Cálculo do Último Host Disponível em uma Rede IP**

Função	`calculate_last_host`
Descrição	Calcula o último host disponível em uma rede IP dado o endereço de *broadcast*.
Exemplo	`char* last_host = calculate_last_host("192.168.3.255"); printf("Último host disponível: %s\n", last_host);`
Saída do Exemplo	`Último host disponível: 192.168.3.254`

Segundo Exemplo	`char* last_host = calculate_last_host( calculate_broadcast_address("192.168.3.54", 24)); printf("Último host disponível: %s\n", last_host);`
Saída do Segundo Exemplo	`Último host disponível: 192.168.3.254`

Fonte: elaborado pelo autor (2024)

Tabela 19 – **Notação CIDR para Máscara de Sub-Rede (IPv6)**

Função	`cidr_to_mask_ipv6`
Descrição	Converte da notação CIDR para uma máscara de sub-rede IPv6.
Exemplo	`char* mask = cidr_to_mask(24); printf("Máscara (IPv6): %s\n", mask);`
Saída do Exemplo	`Máscara (IPv6): ffff:ff00::`

Fonte: elaborado pelo autor (2024)

Tabela 20 – **Máscara de Sub-Rede para Notação CIDR (IPv6)**

Função	`mask_to_cidr_ipv6`
Descrição	Converte uma máscara de sub-rede IPv6 para a notação CIDR.
Exemplo	`int cidr = mask_to_cidr("ffff:ff00::"); printf("CIDR (IPv6): %d\n", cidr);`
Saída do Exemplo	`CIDR (IPv6): 24`

Fonte: elaborado pelo autor (2024)

Tabela 21 – **Cálculo de Endereço de Rede (IPv6)**

Função	`calculate_network_address_ipv6`
Descrição	Calcula o endereço de rede a partir do endereço e a máscara de sub-rede IPv6.
Exemplo	`char* network_address = calculate_network_address_ipv6( "2001:0db8:85a3:0000:0000:8a2e:0370:7334", 16); printf("Endereço de rede (IPv6): %s\n", network_address);`
Saída do Exemplo	`Endereço de rede (IPv6): 2001::`

Fonte: elaborado pelo autor (2024)

Tabela 22 – **Cálculo de Endereço de Broadcast (IPv6)**

Função	`calculate_broadcast_address_ipv6`
Descrição	Calcula o endereço de *broadcast* a partir do endereço e a máscara de sub-rede IPv6.
Exemplo	`char* broadcast_address = calculate_broadcast_address_ipv6( "2001:0db8:85a3:0000:0000:8a2e:0370:7334", 16); printf("Endereço de broadcast (IPv6):\n%s\n", broadcast_address);`
Saída do Exemplo	`Endereço de broadcast (IPv6): 2001:ffff:ffff:ffff:ffff:ffff:ffff:ffff`

Fonte: elaborado pelo autor (2024)

Tabela 23 – **Cálculo do Número de Hosts (IPv6)**

Função	`calculate_number_of_hosts_ipv6`

Descrição	Calcula o número de hosts disponíveis em uma rede IPv6 dada a notação CIDR.
Exemplo	`char* number_of_hosts = calculate_number_of_hosts_ipv6(24); printf("Número de hosts disponíveisc:\n%s\n", number_of_hosts);`
Saída do Exemplo	`Número de hosts disponíveis (IPv6): 20282409603651670423947251286016`

Fonte: elaborado pelo autor (2024)

Tabela 24 – **Cálculo do Primeiro Host Disponível em uma Rede IP (IPv6)**

Função	`calculate_first_host_ipv6`
Descrição	Calcula o primeiro host disponível em uma rede IP dado o endereço de rede.
Exemplo	`char* first_host = calculate_first_host_ipv6("2001:b00::"); printf("Primeiro host disponível (IPv6): %s\n", first_host);`
Saída do Exemplo	`Primeiro host disponível (IPv6): 2001:b00::1`
Segundo Exemplo	`char* first_host = calculate_first_host_ipv6( calculate_network_address_ipv6("2001:0b83::36fb" , 24)); printf("Primeiro host disponível (IPv6): %s\n", first_host);`
Saída do Segundo Exemplo	`Primeiro host disponível (IPv6): 2001:b00::1`

Fonte: elaborado pelo autor (2024)

Tabela 25 – **Cálculo do Último Host Disponível em uma Rede IP (IPv6)**

Função	`calculate_last_host_ipv6`

Descrição	Calcula o último host disponível em uma rede IP dado o endereço de *broadcast*.
Exemplo	`char* last_host = calculate_last_host_ipv6("2001:bff:ffff:ffff:ffff:ffff:ffff:ffff"); printf("Último host disponível (IPv6):\n%s\n", last_host);`
Saída do Exemplo	`Último host disponível (IPv6): 2001:bff:ffff:ffff:ffff:ffff:ffff:fffe`
Segundo Exemplo	`char* last_host = calculate_last_host_ipv6( calculate_broadcast_address_ipv6("2001:0b83::36fb", 24)); printf("Último host disponível (IPv6):\n%s\n", last_host);`
Saída do Segundo Exemplo	`Último host disponível (IPv6): 2001:bff:ffff:ffff:ffff:ffff:ffff:fffe`

Fonte: elaborado pelo autor (2024)

Tabela 26 – **Cálculo de Super-Rede**

Função	`char* calculate_supernet`
Descrição	Calcula a super-rede de uma lista de redes IPv4 fornecidas.
Exemplo	`char* supernet = calculate_supernet(const char* networks[] = { "192.168.1.0", "192.168.1.128" }, 2); printf("Super-rede: %s\n", supernet);`
Saída do Exemplo	`Super-rede: 192.168.1.0`

Fonte: elaborado pelo autor (2024)

Tabela 27 – **Divisão em Sub-Redes**

Função	`char* divide_subnets`

Descrição	Divide uma rede em um número especificado de sub-redes menores.
Exemplo	`char* subnets = divide_subnets("192.168.5.0", 18, 2);` `printf("Sub-redes: %s\n", subnets);`
Saída do Exemplo	`Sub-redes: 192.168.5.0/19 [Available hosts: 8190]` `192.168.37.0/19 [Available hosts: 8190]`

Fonte: elaborado pelo autor (2024)

Tabela 28 – **Adicionar Rota a Tabela de Roteamento**

Função	`char* add_route`
Descrição	Adiciona uma rota a uma tabela de roteamento.
Exemplo	`char* first_route = add_route("192.168.1.0", "255.255.255.0", "192.168.1.1");` `printf("%s\n", first_route);`
Saída do Exemplo	`Route added: 192.168.1.0 via 192.168.1.1 with mask 255.255.255.0`

Fonte: elaborado pelo autor (2024)

Tabela 29 – **Remover Rota da Tabela de Roteamento**

Função	`char* remove_route`
Descrição	Remove uma rota específica da tabela de roteamento.

Exemplo	`char* second_route = add_route("192.168.2.0", "255.255.255.0", "192.168.2.1");`
	`char* remove_second_route = remove_route("192.168.2.0");`
	`printf("%s\n", remove_second_route);`

Saída do Exemplo	`Route to 192.168.2.0 removed`

Fonte: elaborado pelo autor (2024)

Tabela 30 – **Imprimir a Tabela de Roteamento**

Função	`char* print_routes`
Descrição	Imprime todas as rotas da tabela de roteamento.

Exemplo	`char* third_route = add_route("10.0.0.0", "255.0.0.0", "10.0.0.1");`
	`routes_table = print_routes();`
	`printf("%s\n", routes_table);`

Saída do Exemplo	`Routing Table:` `Destination: 10.0.0.0, Mask: 255.0.0.0, Gateway: 10.0.0.1`

Fonte: elaborado pelo autor (2024)

Tabela 31 – **Gateway da Rota**

Função	`const char* find_route`
Descrição	Retorna o *gateway* da primeira rota na tabela que corresponde ao IP dado.

Exemplo	`char* first_route = add_route("192.168.1.0", "255.255.255.0", "192.168.1.1");`
	`const char* gateway = find_route("192.168.1.0");`
	`printf("Gateway: %s\n", gateway);`

Saída do Exemplo	`Gateway: 192.168.1.1`

Fonte: elaborado pelo autor (2024)

Tabela 32 – **Rota com Prefixo Mais Longo**

Função	`int longest_prefix_match`
Descrição	Encontra a rota na tabela com o prefixo mais longo que corresponde ao IP dado.
Exemplo	`char* second_route = add_route( "192.168.2.0", "255.255.255.0","192.168.2.1"); int match_index = longest_prefix_match("192.168.2.54"); printf("Melhor correspondência de rota: %s\n", routing_table[match_index].destination);`
Saída do Exemplo	`Melhor correspondência de rota: 192.168.2.0`

Fonte: elaborado pelo autor (2024)

Tabela 33 – **MAC para IID**

Função	`char* mac_to_iid`
Descrição	Converte um endereço MAC para um identificador de interface IPv6 (IID).
Exemplo	`char* iid = mac_to_iid("AA:BB:CC:DD:EE:FF"); printf("IID: %s\n", iid);`
Saída do Exemplo	`IID: a8bb:ccff:fedd:eeff`

Fonte: elaborado pelo autor (2024)

Tabela 34 – **Cálculo de Checksum IPv4**

Função	`uint16_t calculate_ip_checksum`
Descrição	Calcula o *checksum* IPv4, dado o header.

Exemplo	`uint16_t ip_header[] = {0x4500, 0x0073, 0x0000, 0x4000, 0x4011, 0xb861, 0xc0a8, 0x0001, 0xc0a8, 0x00c7};`
	`uint16_t checksum = calculate_ip_checksum(ip_header, ARRAY_SIZE(ip_header));`
	`printf("Checksum: 0x%04X\n", checksum);`

Saída do Exemplo	`Checksum: 0xB861`

Fonte: elaborado pelo autor (2024)

Tabela 35 – **Verificação de Checksum IPv4**

Função	`bool verify_ip_checksum`
Descrição	Verifica se o *checksum* IPv4 é válido, dado o header.

Exemplo	`uint16_t ip_header[] = {0x4500, 0x0073, 0x0000, 0x4000, 0x4011, 0xb861, 0xc0a8, 0x0001, 0xc0a8, 0x00c7};`
	`bool is_valid = verify_ip_checksum(ip_header, ARRAY_SIZE(ip_header));`
	`printf("O Checksum é válido? %s\n", is_valid ? "Sim." : "Não.");`

Saída do Exemplo	`O Checksum é válido? Sim.`

Fonte: elaborado pelo autor (2024)

Tabela 36 – **Priorização de Pacotes**

Função	`int classify_packet_priority`
Descrição	Classifica a prioridade de um pacote baseado em seu conteúdo.

Exemplo	`int packet_priority = classify_packet_priority("URG DATA PRIORITY=HIGH");`
	`printf("Prioridade do primeiro pacote: %d\n", packet_priority);`
	`packet_priority = classify_packet_priority("ACK PRIORITY=MEDIUM");`

	`printf("Prioridade do segundo pacote: %d\n", packet_priority);`
Saída do Exemplo	`Prioridade do primeiro pacote: 1` `Prioridade do segundo pacote: 2`

Fonte: elaborado pelo autor (2024)

Tabela 37 – **Ping**

Função	`void ping`
Descrição	Executa o comando *ping* para um endereço IP.
Exemplo	`ping("172.20.340.129");`

Saída do Exemplo	`PING 172.20.340.129 (172.20.340.129) 56(84) bytes of data.`
	`64 bytes from 172.20.340.129: icmp_seq=1 ttl=64 time=0.011 ms`
	`64 bytes from 172.20.340.129: icmp_seq=2 ttl=64 time=0.044 ms`
	`64 bytes from 172.20.340.129: icmp_seq=3 ttl=64 time=0.044 ms`
	`64 bytes from 172.20.340.129: icmp_seq=4 ttl=64 time=0.045 ms`
	
	`--- 172.20.340.129 ping statistics ---`
	`4 packets transmitted, 4 received, 0% packet loss, time 3085ms`
	`rtt min/avg/max/mdev = 0.011/0.036/0.045/0.014 ms`

Fonte: elaborado pelo autor (2024)

Tabela 38 – **Traceroute**

Função	`void tracert`
Descrição	Executa o comando *traceroute* para um endereço IP.

Exemplo	`tracert("172.20.34.129");`
Saída do Exemplo	`traceroute to 172.20.235.153 (172.20.235.153), 64 hops max` `  1   172.20.235.153  0.001ms  0.001ms  0.001ms`

Fonte: elaborado pelo autor (2024)

Tabela 39 – **Validação de IPv4**

Função	`bool validate_ip`
Descrição	Valida o formato de um endereço IP.
Exemplo	`bool validate = validate_ip("192.1683.0.1");` `printf("O endereço IP é válido? %s\n", validate ? "Sim." : "Não.");`
Saída do Exemplo	`O endereço IP é válido? Não.`

Fonte: elaborado pelo autor (2024)

Tabela 40 – **Validação de IPv6**

Função	`bool validate_ipv6`
Descrição	Valida o formato de um endereço IPv6.
Exemplo	`bool validate = validate_ipv6("2001:0db8:85a3:0000:0000:8a2e:0154:1dfe");` `printf("O endreço IPv6 é válido? %s\n", validate ? "Sim." : "Não.");`
Saída do Exemplo	`O endreço IPv6 é válido? Sim.`

Fonte: elaborado pelo autor (2024)

Tabela 41 – **Validação de CIDR**

Função	`bool validate_cidr`
Descrição	Valida o comprimento de um prefixo CIDR.
Exemplo	`bool validate = validate_cidr(159);` `printf("O prefixo CIDR é válido? %s\n", validate ? "Sim." : "Não.");`
Saída do Exemplo	`O prefixo CIDR é válido? Não.`

Fonte: elaborado pelo autor (2024)

Tabela 42 – **Validação de Máscara de Sub-Rede**

Função	`bool validate_mask`
Descrição	Valida o formato de uma máscara de sub-rede IPv4.
Exemplo	`bool validate = validate_mask("255.255.0.0");` `printf("A Máscara de Sub-rede (IPv4) é válida? %s\n", validate ? "Sim." : "Não.");`
Saída do Exemplo	`A Máscara de Sub-rede (IPv4) é válida? Sim.`

Fonte: elaborado pelo autor (2024)

Tabela 43 – **Validação de endereço MAC**

Função	`bool validate_mac`

Descrição	Valida o formato de um endereço MAC.
Exemplo	`bool validate = validate_mac("AA:B1:CC:47:ED:FF");` `printf("O endreço MAC é válido? %s\n", validate ? "Sim." : "Não.");`
Saída do Exemplo	`O endreço MAC é válido? Sim.`

Fonte: elaborado pelo autor (2024)

4. PONTOS DE DISCUSSÃO

4.1. Resultados dos Testes

Os testes foram realizados utilizando um arquivo de teste denominado main.c, que contém uma série de chamadas às funções da biblioteca, permitindo verificar se cada uma delas opera conforme o esperado. A seguir, detalha-se a estrutura do arquivo de teste, o ambiente de execução e os resultados obtidos.

4.1.1. Estrutura do Arquivo de Teste

O arquivo main.c foi organizado em seções lógicas, cada uma dedicada a um conjunto específico de funcionalidades da biblioteca. As principais seções do arquivo incluem:

4.1.1.1. *Conversões de Base*

Testes para funções que convertem números decimais para binários e hexadecimais, e vice-versa.

4.1.1.2. *Conversões de IP*

Avaliação das funções que convertem endereços IPv4 e IPv6 entre suas representações em formato binário e suas formas padrão.

4.1.1.3. *Operações de CIDR e Máscaras*

Testes para funções que calculam endereços de rede, endereços de broadcast e máscaras associadas a CIDR.

4.1.1.4. *Funções de Roteamento*

Implementação de testes para adicionar, remover e buscar rotas na tabela de roteamento.

4.1.1.5. *Análise de Tráfego e Diagnóstico de Rede*

Avaliação das funções que realizam operações como *ping* e *traceroute*, além da classificação de pacotes.

4.1.1.6. *Validação*

Testes para verificar a validade de endereços IP, IPv6, máscaras e CIDR.

Cada seção contém chamadas às funções correspondentes, seguidas por impressões no console que reportam os resultados das operações realizadas. O uso de mensagens de erro apropriadas garante que falhas sejam facilmente identificáveis durante os testes.

4.1.2. Ambiente de Execução

Os testes foram realizados em dois ambientes distintos:

4.1.2.1. *No Windows*

Utilizando o terminal MinGW64 do MSYS2, o arquivo foi compilado com o comando "`gcc -o main main.c -L. -lnetworkutils -I. -lm -lgmp -lws2_32`", e o executável gerado foi denominado "main.exe".

4.1.2.2. *No Linux*

No ambiente WSL (*Windows Subsystem for Linux*), o arquivo foi compilado com o comando "`gcc -o main main.c -L. -lnetworkutils -Wl,-rpath=. -lm -lgmp`", e o executável gerado foi simplesmente chamado "main" (Hock-Chuan, 2018).

4.1.3. Execução dos Testes

Após a compilação bem-sucedida em ambos os ambientes, os executáveis foram executados diretamente nos respectivos terminais:

4.1.3.1. *MinGW64 pelo MSYS2*

No terminal do MinGW64 pelo MSYS2, o comando "./main.exe" foi utilizado para iniciar os testes para o ambiente Windows.

4.1.3.2. *WSL*

No terminal WSL Ubuntu, o comando "./main" foi utilizado para executar os testes para o ambiente Linux.

4.1.4. Resultados Obtidos

Os resultados dos testes foram satisfatórios em ambas as plataformas, todas as funções testadas apresentaram resultados esperados, demonstrando que as conversões entre formatos numéricos e representações de endereços IP foram realizadas corretamente, as operações relacionadas a CIDR e máscaras funcionaram conforme projetado (permitindo o cálculo preciso de endereços de rede e broadcast), e , além disso, as funcionalidades de roteamento possibilitaram a adição, remoção e busca de rotas sem erros.

As funções de diagnóstico, como *ping* e *traceroute*, também se mostraram eficazes, retornando informações válidas sobre a conectividade com os endereços IP testados. As validações confirmaram a precisão dos endereços IP e IPv6 fornecidos, incluindo tanto casos válidos quanto inválidos, o que reforça a confiabilidade da biblioteca.

Em resumo, a biblioteca *Network Utilities* demonstrou ser robusta e confiável durante os testes realizados em ambas as plataformas. Os resultados positivos indicam que as implementações atendem aos requisitos funcionais estabelecidos no início do

projeto, proporcionando uma base sólida para futuras extensões ou melhorias na biblioteca.

4.2. Aplicação Educacional

As funções da biblioteca *Network Utilities* foram projetadas para facilitar a manipulação e análise de endereços IP, CIDR, roteamento e diagnósticos de rede, tornando-a uma ferramenta valiosa para aplicações educacionais no contexto de redes de computadores. A seguir, exploraremos como essas funções podem ser aplicadas em exercícios complexos frequentemente encontrados em certificações como a "Cisco Certified Network Associate", mais conhecida pela sigla CCNA (ODOM, 2019), destacando a simplicidade e a eficiência que a biblioteca oferece em comparação com as abordagens tradicionais utilizando bibliotecas padrão da linguagem C.

4.2.1. Exemplos Práticos

4.2.1.1. *Exemplo 1*

Um exercício comum no CCNA envolve a conversão de endereços IP entre decimal e binário. A biblioteca facilita essa tarefa com funções dedicadas:

```
#include "network_utils.h"
#include <stdio.h>

int main() {
    const char* ip = "192.168.1.1";
    char* binary_ip = ip_to_binary(ip);

    if (binary_ip) {
        printf("IP: %s\nEm binário: %s\n",
ip, binary_ip);
        free(binary_ip);
    }
}
```

Em contrapartida, o mesmo código seria resolvido apenas com o uso de bibliotecas padrão da linguagem C da seguinte forma:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>

int main() {
    const char* ip = "192.168.1.1";
    struct in_addr addr;

    unsigned long decimal_ip = ntohl(addr.s_addr);

    char binary_ip[33];
```

```
    binary_ip[32] = '\0';

    for (int i = 31; i >= 0; i--) {
        binary_ip[i] = (decimal_ip % 2) ? '1' : '0';
        decimal_ip /= 2;
    }

    printf("IP:    %s\nEm    binário:    %s\n",    ip,
binary_ip);

    return EXIT_SUCCESS;
}
```

4.2.1.2. *Exemplo 2*

Em um exercício onde os alunos devem gerenciar uma tabela de roteamento, as funções da biblioteca tornam o processo mais intuitivo:

```
#include "network_utils.h"
#include <stdio.h>

int main() {
    add_route("192.168.1.0", "255.255.255.0",
"192.168.1.1");

    char* routes = print_routes();
    printf("Tabela de Roteamento:\n%s\n", routes);

    const char* gateway =
find_route("192.168.1.5");

    if (gateway) {
        printf("A rota para 192.168.1.5 tem o
gateway: %s\n", gateway);
    } else {
        printf("Nenhuma rota encontrada para
192.168.1.5.\n");
    }

    free(routes);
}
```

Em contrapartida, o mesmo código seria resolvido apenas com o uso de bibliotecas padrão da linguagem C da seguinte forma:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ROUTES 100

typedef struct {
    char destination[16];
    char mask[16];
    char gateway[16];
} Route;
```

```
Route routing_table[MAX_ROUTES];
int num_routes = 0;

int main() {
    const char* first_destination = "192.168.1.0";
    const char* first_mask = "255.255.255.0";
    const char* first_gateway = "192.168.1.1";

    if (num_routes < MAX_ROUTES) {
strncpy(routing_table[num_routes].destination,
first_destination,
sizeof(routing_table[num_routes].destination));
strncpy(routing_table[num_routes].mask, first_mask,
sizeof(routing_table[num_routes].mask));
strncpy(routing_table[num_routes].gateway,
first_gateway,
sizeof(routing_table[num_routes].gateway));
        num_routes++;
printf("Rota adicionada: %s | Máscara: %s |
Gateway: %s\n", first_destination, first_mask,
first_gateway);
    } else {
        printf("Tabela de roteamento cheia.\n");
    }

    printf("\nTabela de Roteamento:\n");
    for (int i = 0; i < num_routes; i++) {
        printf("Destino: %s | Máscara: %s |
Gateway: %s\n",
               routing_table[i].destination,
               routing_table[i].mask,
               routing_table[i].gateway);
    }

    const char* ip_to_find = "192.168.1.5";
    int found_route = -1;


    for (int i = 0; i < num_routes; i++) {
        if (strcmp(routing_table[i].destination,
ip_to_find) == 0) {
            found_route = i;
            break;
        }
    }


    if (found_route != -1) {
printf("A rota para %s tem o gateway: %s\n",
ip_to_find, routing_table[found_route].gateway);
    } else {
        printf("Nenhuma rota encontrada para %s\n",
ip_to_find);
    }

    return 0;
}
```

Neste exemplo, a implementação manual da tabela de roteamento requer várias etapas para adicionar rotas e imprimir a tabela. Além disso, a busca por rotas é feita através de comparações manuais entre *strings*.

4.2.1.3. *Exemplo 3*

Dado um exercício de cálculo do endereço de broadcast tendo sido fornecido o endereço de rede, a resolução com as funções da biblioteca seguiria da seguinte forma:

```
#include <stdio.h>
#include <stdlib.h>
#include "network_utils.h"

int main() {
    const char* ip = "172.16.16.0";
    int cidr = 22;

char* broadcast_address =
calculate_broadcast_address(ip, cidr);

    if (broadcast_address) {
printf("Endereço de Broadcast para %s/%d: %s\n",
ip, cidr, broadcast_address);
        free(broadcast_address);
    } else {
fprintf(stderr, "Erro ao calcular o endereço de
broadcast.\n");
    }

    return 0;
}
```

Em contrapartida, o mesmo código seria resolvido apenas com o uso de bibliotecas padrão da linguagem C da seguinte forma:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>

int main() {
    const char* ip = "172.16.16.0";
    int cidr = 22;
    struct in_addr ip_addr, mask_addr, broadcast_addr;

    if (inet_pton(AF_INET, ip, &ip_addr) != 1) {
        fprintf(stderr, "Erro ao converter o IP.\n");
        return EXIT_FAILURE;
    }

unsigned int mask = (cidr == 0) ? 0 : ~((1 << (32 -
cidr)) - 1);
    mask_addr.s_addr = htonl(mask);
```

```
broadcast_addr.s_addr = (ip_addr.s_addr &
mask_addr.s_addr) | ~mask_addr.s_addr;

    char broadcast_str[INET_ADDRSTRLEN];
inet_ntop(AF_INET, &broadcast_addr, broadcast_str,
INET_ADDRSTRLEN);

printf("Endereço de Broadcast para %s/%d: %s\n", ip,
cidr, broadcast_str);

    return EXIT_SUCCESS;
}
```

A comparação entre as duas abordagens destaca como a biblioteca Network Utilities simplifica significativamente tarefas comuns em redes de computadores em comparação com implementações manuais utilizando apenas funções padrão da linguagem C. A utilização da biblioteca não só melhora a eficiência do desenvolvimento como também aumenta a clareza e reduz o potencial para erros no código final.

5. CONCLUSÃO

A biblioteca *Network Utilities*, desenvolvida neste projeto, representa uma contribuição importante para a manipulação e análise de endereços IP, CIDR, roteamento e diagnósticos de rede, especialmente no contexto educacional. Sua concepção e implementação foram desafiadoras, especialmente devido à necessidade de garantir compatibilidade entre sistemas operacionais Linux e Windows, e de atender a requisitos educacionais específicos, promovendo um aprendizado prático e acessível. Desde o início, a busca por bibliotecas equivalentes que funcionassem em ambos os sistemas se revelou uma tarefa árdua, já que muitas soluções existentes são limitadas a ambientes específicos, ou até possuem dependências complexas que dificultam sua utilização em contextos acadêmicos, esse desafio reforçou a necessidade de criar uma solução própria, capaz de atender a múltiplas plataformas e facilitar o ensino dos conceitos de redes de computadores.

Durante o desenvolvimento, um dos aspectos mais exigentes foi a gestão de memória, especialmente no arquivo de teste “main.c”. A necessidade de liberar adequadamente a memória alocada dinamicamente para evitar vazamentos foi um ponto crítico, exigindo uma abordagem rigorosa para garantir que cada recurso fosse liberado após o uso. Esse processo não apenas garantiu a estabilidade da biblioteca, mas também serviu como uma oportunidade de aprendizado sobre boas práticas de programação em C, especialmente em um ambiente de baixo nível.

Os testes realizados confirmaram que a biblioteca atende aos requisitos funcionais estabelecidos, permitindo que usuários e desenvolvedores realizem operações complexas, como cálculos de endereços de rede e broadcast, conversões entre bases numéricas e manipulações de máscaras de sub-rede, de forma eficiente e intuitiva. A modularidade do código, um dos principais pontos fortes da biblioteca, permite que as funções sejam reutilizadas em diferentes contextos, promovendo uma organização clara e facilitando futuras expansões.

Entretanto, algumas limitações foram identificadas. A biblioteca, em sua versão atual, lida principalmente com operações básicas e intermediárias relacionadas a IPv4 e IPv6, para aplicações mais complexas ou em larga escala será necessário expandir as funcionalidades para incluir suporte a protocolos adicionais, como OSPF ou BGP, e métodos avançados de roteamento; além disso, o tratamento de erros, embora funcional, pode ser aprimorado para lidar com cenários mais variados, proporcionando maior robustez ao código; documentação também representa uma área a ser melhorada, pois exemplos mais detalhados e contextualizados poderiam facilitar a adoção da biblioteca por novos usuários; e ainda, a validação de entradas, embora implementada, poderia ser refinada para evitar falhas em tempo de execução devido a dados malformados.

Apesar dessas limitações, as perspectivas futuras para a biblioteca são promissoras, visto que com a inclusão de funcionalidades adicionais, como suporte a redes definidas por software (SDN) e virtualização de redes, bem como a criação de uma interface mais amigável para desenvolvedores iniciantes, pode expandir significativamente seu impacto. O desenvolvimento de testes automatizados e a integração com ferramentas externas também são incrementos que podem elevar o nível de confiabilidade e funcionalidade da biblioteca. Além disso, a colaboração com outros desenvolvedores e a formação de uma comunidade em torno da ferramenta podem gerar inovações que atendam a novas demandas do setor de redes.

Em síntese, o desenvolvimento da *Network Utilities* foi um processo enriquecedor, que não apenas resultou em uma solução prática para problemas comuns em redes de computadores, mas também estabeleceu uma base sólida para futuras inovações. O projeto demonstrou que, mesmo diante de desafios como compatibilidade entre sistemas e gestão de memória, é possível criar ferramentas robustas e eficazes. Com melhorias contínuas e expansões planejadas, a biblioteca tem o potencial de se tornar uma referência valiosa para profissionais da área e estudantes interessados em aprofundar seus conhecimentos em redes de computadores.

6. REFERÊNCIAS BIBLIOGRÁFICAS

BHATTI, M. K. L.; KHAN, I.; KIM, K.-I. ***A comprehensive review of Internet of Things: Technology stack, middlewares, and fog/edge computing interface***. Sensors, v. 22, n. 3, p. 995, 2022. DOI: 10.3390/s22030995.

COMPUTER HISTORY MUSEUM. **Dennis Ritchie**. 2024. Disponível em: https://www.computerhistory.org/profile/dennis-ritchie/. Acesso em: 21 nov. 2024.

CROCKER, Stephen D. ***Arpanet and Its Evolution — A Report Card***. IEEE Communications Magazine, v. 59, n. 12, p. 118–124, dez. 2021. Disponível em: https://ieeexplore.ieee.org/document/9681633. Acesso em: 21 nov. 2024.

FREE SOFTWARE FOUNDATION, INC. ***GCC 14.2 Manual***. Disponível em: <https://gcc.gnu.org/onlinedocs/gcc-14.2.0/gcc/>. Acesso em: 21 nov. 2024.

GUDHKA, Drashti. ***Computer Network***. International Journal for Research in Applied Science and Engineering Technology (IJRASET), 2024. Disponível em: https://doi.org/10.22214/ijraset.2024.57862. Acesso em: 21 nov. 2024.

HEAD. ***Netzwerktopologie Bus***. Wikimedia Commons, 2023.

HOCK-CHUAN, Chua. ***GCC and Make - Compiling, Linking and Building C/C++ Applications***. Nanyang Technological University, 2018. Disponível em: https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html. Acesso em: 21 nov. 2024.

HOWARD, Michael. ***Router & Switch Survey: Cisco, Juniper, Huawei & Nokia form Top Tier***. IEEE Communications Society Technology Blog, 2017. Disponível em: https://techblog.comsoc.org/. Acesso em: 21 nov. 2024.

KOMAN90. ***Diagram Of A True Mesh Topology Network***. Wikimedia Commons, 2009.

MSYS2. ***What is MSYS2?*** Disponível em: <https://www.msys2.org/docs/what-is-msys2/>. Acesso em: 21 nov. 2024.

ODOM, Wendell. ***Exam Profile: Cisco 200-301 CCNA***. Pearson IT Certification, 2019. Disponível em: https://www.pearsonitcertification.com/articles/article.aspx?p=2982442. Acesso em: 21 nov. 2024

QEEF. ***Čeština: Hvězdicová topologie***. Wikimedia Commons, 2015.

RITCHIE, D. M. ***The development of the C language***. HOPL-II: The second ACM SIGPLAN conference on History of programming languages, v. 28, n. 3, p. 201–208, 1 jan. 1993.

ROMERO, Victor; et al. ***Mingw-w64***. Microsoft, 2024. Disponível em: https://learn.microsoft.com/en-us/vcpkg/users/platforms/mingw. Acesso em: 21 nov. 2024.

ROONEY, T.; DOOLEY, M. ***IP Address Management***. 2. ed. [s.l.] Wiley-IEEE Press, 2021. p. 31–50.

SIMEONOVSKI, Kiril. ***Ring network scheme***. Wikimedia Commons, 2017.

TANENBAUM, Andrew S.; WETHERALL, David J. Redes de computadores. 5. ed. São Paulo: Pearson Prentice Hall, 2011

WOJCIAKOWSKI, Matt; et al. ***Windows Subsystem for Linux Documentation***. Microsoft, 2022. Disponível em: https://learn.microsoft.com/en-us/windows/wsl/. Acesso em: 21 nov. 2024.

ZIKRIA, Yousaf B; et al. ***Next-generation Internet of Things (IoT): Opportunities, challenges, and solutions***. MDPI Electronics, 2022. DOI: 10.3390/s21041174.

7. APÊNDICES

7.1. APÊNDICE A

Código do arquivo cabeçalho da biblioteca “network_utils.h”:

```
#ifndef NETWORK_UTILS_H
#define NETWORK_UTILS_H

#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>

// Funções de conversão IPv4, IPv6 e resumo de IPv6
char* decimal_to_binary(int decimal);
int binary_to_decimal(const char* binary_str);
char* hexadecimal_to_binary(const char* hex_str);
char* binary_to_hexadecimal(const char* binary_str);
char* decimal_to_hexadecimal(int decimal);
int hexadecimal_to_decimal(const char* hex_str);
char* ip_to_binary(const char* ip);
char* binary_to_ip(const char* binary_str);
char* ipv6_to_binary(const char* ipv6);
char* binary_to_ipv6(const char* binary_str);
char* summarize_ipv6(const char* ipv6);

// Operações de CIDR e máscaras
char* cidr_to_mask(int cidr);
int mask_to_cidr(const char* mask);
char* calculate_network_address(const char* ip, int cidr);
char* calculate_broadcast_address(const char* ip, int cidr);
int calculate_number_of_hosts(int cidr);
char* calculate_first_host(const char* network_ip);
char* calculate_last_host(const char* broadcast_ip);
char* cidr_to_mask_ipv6(int cidr);
int mask_to_cidr_ipv6(const char* mask);
char* calculate_network_address_ipv6(const char* ip, int cidr);
char* calculate_broadcast_address_ipv6(const char* ip, int cidr);
char* calculate_number_of_hosts_ipv6(int cidr);
char* calculate_first_host_ipv6(const char* network_ip);
char* calculate_last_host_ipv6(const char* broadcast_ip);

// Funções de super-rede e sub-rede
char* calculate_supernet(const char** networks, int num_networks);
```

```
char* divide_subnets(const char* network_ip, int cidr, int num_subnets);

// Funções de roteamento
typedef struct {
    char destination[16];
    char mask[16];
    char gateway[16];
} Route;

extern Route* routing_table;
extern int num_routes;

void resize_routing_table(int new_size);
char* add_route(const char* destination, const char* mask, const char*
gateway);
char* remove_route(const char* destination);
char* print_routes();
const char* find_route(const char* ip);
int longest_prefix_match(const char* ip);

// Funções de análise de tráfego
#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))

char* mac_to_iid(const char* mac);
uint16_t calculate_ip_checksum(const uint16_t pairs[], size_t num_pairs);
bool verify_ip_checksum(const uint16_t pairs[], size_t num_pairs);
int classify_packet_priority(const char* packet);

// Funções de diagnóstico de rede
void ping(const char* ip);
void tracert(const char* ip);

// Funções de validação
bool validate_ip(const char* ip);
bool validate_ipv6(const char* ipv6);
bool validate_cidr(int prefix_length);
bool validate_mask(const char* mask);
bool validate_mac(const char* mac);


#endif // NETWORK_UTILS_H
```

7.2. APÊNDICE B

Código do arquivo de implantação das funções da biblioteca “network_utils.c”:

```
#include "network_utils.h"
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>
#include <gmp.h>

#ifdef _WIN32
#include <winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "ws2_32.lib")
#else
#include <arpa/inet.h>
#endif

#ifdef _WIN32
int ipv4_pton(int af, const char *src, void *dst) {
    return InetPtonA(af, src, dst);
}

const char *ipv4_ntop(int af, const void *src, char *dst, size_t size) {
    return InetNtopA(af, src, dst, size);
}
#else
#define ipv4_pton inet_pton
#define ipv4_ntop inet_ntop
#endif


// Função auxiliar para alocar strings

char* allocate_string(int size) {
    char* str = (char*)malloc(size);
    if (!str) {
        fprintf(stderr, "Erro de alocação de memória\n");
        exit(EXIT_FAILURE);
    }
    return str;
}

// --------- Funções de Conversão IPv4, IPv6 e resumo de IPv6 ---------
```

```
char* decimal_to_binary(int decimal) {
    if (decimal == 0) {
        char* zero = allocate_string(2);
        zero[0] = '0';
        zero[1] = '\0';
        return zero;
    }

    char* binary = allocate_string(33);
    int start = -1;

    for (int i = 31; i >= 0; i--) {
        binary[31 - i] = (decimal & (1 << i)) ? '1' : '0';
        if (binary[31 - i] == '1' && start == -1) {
            start = 31 - i;
        }
    }

    binary[32] = '\0';

    char* trimmed_binary = allocate_string(33 - start);
    for (int i = 0; i < 33 - start; i++) {
        trimmed_binary[i] = binary[start + i];
    }

    free(binary);
    return trimmed_binary;
}

int binary_to_decimal(const char* binary_str) {
    int decimal = 0;
    for (int i = 0; binary_str[i] != '\0'; i++) {
        decimal = (decimal << 1) | (binary_str[i] - '0');
    }
    return decimal;
}

char* hexadecimal_to_binary(const char* hex_str) {
    int decimal = hexadecimal_to_decimal(hex_str);
    return decimal_to_binary(decimal);
}

char* binary_to_hexadecimal(const char* binary_str) {
    int decimal = binary_to_decimal(binary_str);
    return decimal_to_hexadecimal(decimal);
```

```
}

char* decimal_to_hexadecimal(int decimal) {
    char* hex = allocate_string(9);
    snprintf(hex, 9, "%X", decimal);
    return hex;
}

int hexadecimal_to_decimal(const char* hex_str) {
    int decimal;
    sscanf(hex_str, "%X", &decimal);
    return decimal;
}

char* ip_to_binary(const char* ip) {
    struct in_addr addr;
    ipv4_pton(AF_INET, ip, &addr);
    return decimal_to_binary(ntohl(addr.s_addr));
}

char* binary_to_ip(const char* binary_str) {
    int decimal = binary_to_decimal(binary_str);
    struct in_addr addr;
    addr.s_addr = htonl(decimal);
    char* ip = allocate_string(INET_ADDRSTRLEN);
    ipv4_ntop(AF_INET, &addr, ip, INET_ADDRSTRLEN);
    return ip;
}

char* ipv6_to_binary(const char* ipv6) {
    struct in6_addr addr;

    if (ipv4_pton(AF_INET6, ipv6, &addr) != 1) {
        return NULL;
    }

    char* binary_str = allocate_string(129);

    for (int i = 0; i < 16; i++) {
        for (int j = 7; j >= 0; j--) {
            binary_str[i * 8 + (7 - j)] = (addr.s6_addr[i] & (1 << j)) ?
'1' : '0';
        }
    }
```

```
    binary_str[128] = '\0';
    return binary_str;
}

char* binary_to_ipv6(const char* binary_str) {
    if (strlen(binary_str) != 128) {
        return NULL;
    }

    struct in6_addr addr = {0};

    for (int i = 0; i < 16; i++) {
        uint8_t byte = 0;
        for (int j = 0; j < 8; j++) {
            byte = (byte << 1) | (binary_str[i * 8 + j] - '0');
        }
        addr.s6_addr[i] = byte;
    }

    char* ipv6_str = allocate_string(INET6_ADDRSTRLEN);

    if (ipv4_ntop(AF_INET6, &addr, ipv6_str, INET6_ADDRSTRLEN) == NULL)
{
        free(ipv6_str);
        return NULL;
    }

    return ipv6_str;
}

char* summarize_ipv6(const char* ipv6) {
    char block[8][5] = {0};
    int position = 0;

    char* ipv6_summarized = (char*)malloc(40 * sizeof(char));
    if (!ipv6_summarized) {
        return NULL;
    }
    ipv6_summarized[0] = '\0';

    for (int i = 0; i < 8; i++) {
        int block_position = 0;
        while (ipv6[position] != ':' && ipv6[position] != '\0') {
            block[i][block_position++] = ipv6[position++];
        }
```

```
        block[i][block_position] = '\0';
        position++;
    }

    for (int i = 0; i < 8; i++) {
        int j = 0;
        while (block[i][j] == '0' && j < strlen(block[i]) - 1) {
            j++;
        }
        memmove(block[i], &block[i][j], strlen(block[i]) - j + 1);
    }

    int compress = 0;
    for (int i = 0; i < 8; i++) {
        if (strcmp(block[i], "0") == 0 && !compress) {
            strcat(ipv6_summarized, ":");
            compress = 1;
            while (i < 8 && strcmp(block[i], "0") == 0) i++;
            i--;
        } else {
            strcat(ipv6_summarized, block[i]);
            if (i < 7) strcat(ipv6_summarized, ":");
            compress = 0;
        }
    }

    if (ipv6_summarized[0] == ':' && ipv6_summarized[1] == ':') {
        memmove(ipv6_summarized,       ipv6_summarized       +       1,
strlen(ipv6_summarized));
    }
    if   (ipv6_summarized[strlen(ipv6_summarized)   -   1]   ==   ':'   &&
ipv6_summarized[strlen(ipv6_summarized) - 2] == ':') {
        ipv6_summarized[strlen(ipv6_summarized) - 1] = '\0';
    }

    return ipv6_summarized;
}

// --------- Funções CIDR e Máscaras ---------

char* cidr_to_mask(int cidr) {
    if (cidr < 0 || cidr > 32) {
        return NULL;
    }
```

```
    char* mask = malloc(16);
    if (!mask) {
        return NULL;
    }

    unsigned int mask_value = 0xFFFFFFFF << (32 - cidr);

    snprintf(mask, 16, "%d.%d.%d.%d",
             (mask_value >> 24) & 0xFF,
             (mask_value >> 16) & 0xFF,
             (mask_value >> 8) & 0xFF,
             mask_value & 0xFF);

    return mask;
}

int mask_to_cidr(const char* mask) {

    int octets[4];
    if (sscanf(mask, "%d.%d.%d.%d", &octets[0], &octets[1], &octets[2],
&octets[3]) != 4) {
        return -1;
    }

    int cidr = 0;
    for (int i = 0; i < 4; i++) {

        while (octets[i] > 0) {
            cidr += (octets[i] & 1);
            octets[i] >>= 1;
        }
    }
    return cidr;
}

char* calculate_network_address(const char* ip, int cidr) {
    struct in_addr ip_addr, mask_addr;

    ipv4_pton(AF_INET, ip, &ip_addr);

    char* mask_str = cidr_to_mask(cidr);
    if (!mask_str) {
        return NULL;
    }
```

```
    ipv4_pton(AF_INET, mask_str, &mask_addr);
    free(mask_str);

    ip_addr.s_addr &= mask_addr.s_addr;

    char* network_ip = allocate_string(INET_ADDRSTRLEN);
    if (!network_ip) {
        return NULL;
    }

    ipv4_ntop(AF_INET, &ip_addr, network_ip, INET_ADDRSTRLEN);
    return network_ip;
}

char* calculate_broadcast_address(const char* ip, int cidr) {
    struct in_addr ip_addr, mask_addr;

    ipv4_pton(AF_INET, ip, &ip_addr);

    char* mask_str = cidr_to_mask(cidr);
    if (!mask_str) {
        return NULL;
    }

    ipv4_pton(AF_INET, mask_str, &mask_addr);
    free(mask_str);

    ip_addr.s_addr |= ~mask_addr.s_addr;

    char* broadcast_ip = allocate_string(INET_ADDRSTRLEN);
    if (!broadcast_ip) {
        return NULL;
    }

    ipv4_ntop(AF_INET, &ip_addr, broadcast_ip, INET_ADDRSTRLEN);
    return broadcast_ip;
}

int calculate_number_of_hosts(int cidr) {
    return (1 << (32 - cidr)) - 2;
}

char* calculate_first_host(const char* network_ip) {

    char* first_host = malloc(16);
```

```
    if (!first_host) {
        return NULL;
    }

    int octets[4];
    sscanf(network_ip, "%d.%d.%d.%d", &octets[0], &octets[1], &octets[2],
&octets[3]);

    if (octets[3] < 255) {
        octets[3] += 1;
    } else {

        octets[3] = 0;
        if (octets[2] < 255) {
            octets[2] += 1;
        } else {
            octets[2] = 0;
            if (octets[1] < 255) {
                octets[1] += 1;
            } else {
                octets[1] = 0;
                octets[0] += 1;
            }
        }
    }

    snprintf(first_host,   16,   "%d.%d.%d.%d",   octets[0],   octets[1],
octets[2], octets[3]);
    return first_host;
}

char* calculate_last_host(const char* broadcast_ip) {

    char* last_host = malloc(16);
    if (!last_host) {
        return NULL;
    }

    int octets[4];
    sscanf(broadcast_ip,    "%d.%d.%d.%d",    &octets[0],    &octets[1],
&octets[2], &octets[3]);

    if (octets[3] > 0) {
        octets[3] -= 1;
    } else {
```

```
        octets[3] = 255;
        if (octets[2] > 0) {
            octets[2] -= 1;
        } else {
            octets[2] = 255;
            if (octets[1] > 0) {
                octets[1] -= 1;
            } else {
                octets[1] = 255;
                if (octets[0] > 0) {
                    octets[0] -= 1;
                } else {
                    free(last_host);
                    return NULL;
                }
            }
        }
    }

    snprintf(last_host,   16,   "%d.%d.%d.%d",   octets[0],   octets[1],
octets[2], octets[3]);
    return last_host;
}

char* cidr_to_mask_ipv6(int cidr) {
    struct in6_addr mask_addr = {0};
    for (int i = 0; i < cidr / 8; i++) {
        mask_addr.s6_addr[i] = 0xFF;
    }
    if (cidr % 8) {
        mask_addr.s6_addr[cidr / 8] = (0xFF << (8 - (cidr % 8)));
    }

    char* mask = malloc(INET6_ADDRSTRLEN);
    if     (!mask     ||     !ipv4_ntop(AF_INET6,     &mask_addr,     mask,
INET6_ADDRSTRLEN)) {
        free(mask);
        return NULL;
    }
    return mask;
}

int mask_to_cidr_ipv6(const char* mask) {
    struct in6_addr mask_addr;
    if (ipv4_pton(AF_INET6, mask, &mask_addr) <= 0) {
```

```
        return -1;
    }

    int cidr = 0;
    for (int i = 0; i < 16; i++) {
        unsigned char byte = mask_addr.s6_addr[i];
        while (byte) {
            cidr += byte & 1;
            byte >>= 1;
        }
    }
    return cidr;
}

char* calculate_network_address_ipv6(const char* ip, int cidr) {
    struct in6_addr addr;
    struct in6_addr netmask;
    struct in6_addr network;

    if (ipv4_pton(AF_INET6, ip, &addr) <= 0) return NULL;

    memset(&netmask, 0, sizeof(netmask));
    for (int i = 0; i < cidr / 8; i++) {
        netmask.s6_addr[i] = 0xFF;
    }
    if (cidr % 8) {
        netmask.s6_addr[cidr / 8] = (0xFF << (8 - (cidr % 8)));
    }

    for (int i = 0; i < 16; i++) {
        network.s6_addr[i] = addr.s6_addr[i] & netmask.s6_addr[i];
    }

    char* network_ip = malloc(INET6_ADDRSTRLEN);
    if (!network_ip || !ipv4_ntop(AF_INET6, &network, network_ip,
INET6_ADDRSTRLEN)) {
        free(network_ip);
        return NULL;
    }

    return network_ip;
}

char* calculate_broadcast_address_ipv6(const char* ip, int cidr) {
    struct in6_addr addr;
```

```
    struct in6_addr netmask;
    struct in6_addr broadcast;

    if (ipv4_pton(AF_INET6, ip, &addr) <= 0) return NULL;

    memset(&netmask, 0, sizeof(netmask));
    for (int i = 0; i < cidr / 8; i++) {
        netmask.s6_addr[i] = 0xFF;
    }
    if (cidr % 8) {
        netmask.s6_addr[cidr / 8] = (0xFF << (8 - (cidr % 8)));
    }

    for (int i = 0; i < 16; i++) {
        broadcast.s6_addr[i] = addr.s6_addr[i] | ~netmask.s6_addr[i];
    }

    char* broadcast_ip = malloc(INET6_ADDRSTRLEN);
    if (!broadcast_ip || !ipv4_ntop(AF_INET6, &broadcast, broadcast_ip,
INET6_ADDRSTRLEN)) {
        free(broadcast_ip);
        return NULL;
    }

    return broadcast_ip;
}

char* calculate_number_of_hosts_ipv6(int cidr) {
    if (cidr < 0 || cidr > 128) {
        return NULL;
    }

    mpz_t num_hosts;
    mpz_init(num_hosts);

    mpz_ui_pow_ui(num_hosts, 2, 128 - cidr);

    char* result_str = mpz_get_str(NULL, 10, num_hosts);

    mpz_clear(num_hosts);

    return result_str;
}

char* calculate_first_host_ipv6(const char* network_ip) {
```

```
    struct in6_addr addr;
    if (ipv4_pton(AF_INET6, network_ip, &addr) <= 0) return NULL;

    addr.s6_addr[15] += 1;

    char* first_host_ip = malloc(INET6_ADDRSTRLEN);
    if (!first_host_ip || !ipv4_ntop(AF_INET6, &addr, first_host_ip,
INET6_ADDRSTRLEN)) {
        free(first_host_ip);
        return NULL;
    }

    return first_host_ip;
}

char* calculate_last_host_ipv6(const char* broadcast_ip) {
    struct in6_addr addr;
    if (ipv4_pton(AF_INET6, broadcast_ip, &addr) <= 0) return NULL;

    addr.s6_addr[15] -= 1;

    char* last_host_ip = malloc(INET6_ADDRSTRLEN);
    if (!last_host_ip || !ipv4_ntop(AF_INET6, &addr, last_host_ip,
INET6_ADDRSTRLEN)) {
        free(last_host_ip);
        return NULL;
    }

    return last_host_ip;
}


// --------- Funções de Super-rede e Sub-rede ---------

char* calculate_supernet(const char** networks, int num_networks) {
    struct in_addr addr;
    ipv4_pton(AF_INET, networks[0], &addr);
    addr.s_addr &= htonl(~0 << 8);
    char* supernet = allocate_string(INET_ADDRSTRLEN);
    ipv4_ntop(AF_INET, &addr, supernet, INET_ADDRSTRLEN);
    return supernet;
}
```

```
char* divide_subnets(const char* network_ip, int cidr, int num_subnets)
{

    int bits_needed = (int)ceil(log2(num_subnets));
    int new_cidr = cidr + bits_needed;


    if (new_cidr > 30) {
        return NULL;
    }

    int total_subnets = 1 << bits_needed;
    int hosts_per_subnet = (1 << (32 - new_cidr)) - 2;

    size_t entry_size = INET_ADDRSTRLEN + 40;
    size_t buffer_size = entry_size * total_subnets;
    char* result = (char*)malloc(buffer_size);
    if (!result) {
        return NULL;
    }
    result[0] = '\0';

    struct in_addr addr;
    ipv4_pton(AF_INET, network_ip, &addr);

    for (int i = 0; i < total_subnets; i++) {
        struct in_addr subnet_addr = addr;
        subnet_addr.s_addr = ntohl(ntohl(addr.s_addr) + (i * (1 << (32 -
new_cidr))));

        char subnetwork[INET_ADDRSTRLEN];
        ipv4_ntop(AF_INET, &subnet_addr, subnetwork, INET_ADDRSTRLEN);

        char subnet_info[entry_size];
        snprintf(subnet_info,  sizeof(subnet_info),  "%s/%d  [Available
hosts: %d]\n", subnetwork, new_cidr, hosts_per_subnet);
        strncat(result, subnet_info, buffer_size - strlen(result) - 1);
    }

    return result;
}

// --------- Funções de Roteamento ---------

Route* routing_table = NULL;
```

```
int num_routes = 0;

void resize_routing_table(int new_size) {
    routing_table = realloc(routing_table, new_size * sizeof(Route));
    if (routing_table == NULL && new_size > 0) {
        fprintf(stderr, "Erro ao realocar a tabela de rotas.\n");
        exit(1);
    }
}

char* add_route(const char* destination, const char* mask, const char*
gateway) {
    resize_routing_table(num_routes + 1);
    strncpy(routing_table[num_routes].destination,          destination,
sizeof(routing_table[num_routes].destination));
    strncpy(routing_table[num_routes].mask,                       mask,
sizeof(routing_table[num_routes].mask));
    strncpy(routing_table[num_routes].gateway,                  gateway,
sizeof(routing_table[num_routes].gateway));
    num_routes++;

    char* result = malloc(100);
    snprintf(result, 100, "Route added: %s via %s with mask %s\n",
destination, gateway, mask);
    return result;
}

char* remove_route(const char* destination) {
    char* result = malloc(50);
    if (!result) return NULL;

    for (int i = 0; i < num_routes; i++) {
        if (strcmp(routing_table[i].destination, destination) == 0) {
            for (int j = i; j < num_routes - 1; j++) {
                routing_table[j] = routing_table[j + 1];
            }
            num_routes--;
            resize_routing_table(num_routes);

            snprintf(result, 50, "Route to %s removed\n", destination);
            return result;
        }
    }
    snprintf(result, 50, "Route to %s not found\n", destination);
    return result;
```

```
}

char* print_routes() {
    size_t buffer_size = (num_routes * 100) + 30;
    char* result = malloc(buffer_size);
    if (!result) return NULL;

    if (num_routes == 0) {
        snprintf(result, buffer_size, "No routes in the table.\n");
    } else {
        strcpy(result, "Routing Table:\n");
        for (int i = 0; i < num_routes; i++) {
            char route_info[100];
            snprintf(route_info, sizeof(route_info), "Destination: %s,
Mask: %s, Gateway: %s\n",
                     routing_table[i].destination,
                     routing_table[i].mask,
                     routing_table[i].gateway);
            strncat(result, route_info, buffer_size - strlen(result) -
1);
        }
    }
    return result;
}

int longest_prefix_match(const char* ip) {
    int best_match = -1;
    int longest_prefix_length = -1;

    struct in_addr ip_addr, dest_addr, mask_addr;
    if (ipv4_pton(AF_INET, ip, &ip_addr) <= 0) {
        return best_match;
    }

    for (int i = 0; i < num_routes; i++) {
        ipv4_pton(AF_INET, routing_table[i].destination, &dest_addr);
        ipv4_pton(AF_INET, routing_table[i].mask, &mask_addr);

        if ((ip_addr.s_addr & mask_addr.s_addr) == (dest_addr.s_addr &
mask_addr.s_addr)) {
            int prefix_length = __builtin_popcount(mask_addr.s_addr);
            if (prefix_length > longest_prefix_length) {
                longest_prefix_length = prefix_length;
                best_match = i;
            }
```

```
        }
    }
    return best_match;
}

const char* find_route(const char* ip) {
    for (int i = 0; i < num_routes; i++) {
        struct in_addr ip_addr, dest_addr, mask_addr;

        ipv4_pton(AF_INET, ip, &ip_addr);
        ipv4_pton(AF_INET, routing_table[i].destination, &dest_addr);
        ipv4_pton(AF_INET, routing_table[i].mask, &mask_addr);

        if ((ip_addr.s_addr & mask_addr.s_addr) == (dest_addr.s_addr &
mask_addr.s_addr)) {
            return routing_table[i].gateway;
        }
    }
    return NULL;
}

char* mac_to_iid(const char* mac) {
    unsigned int mac_bytes[6];

    sscanf(mac, "%02x:%02x:%02x:%02x:%02x:%02x",
           &mac_bytes[0], &mac_bytes[1], &mac_bytes[2],
           &mac_bytes[3], &mac_bytes[4], &mac_bytes[5]);

    mac_bytes[0] ^= 0x02;

    char* iid = malloc(24);
    if (iid == NULL) {
        return NULL;
    }

    sprintf(iid, "%02x%02x:%02xff:fe%02x:%02x%02x",
            mac_bytes[0], mac_bytes[1], mac_bytes[2],
            mac_bytes[3], mac_bytes[4], mac_bytes[5]);

    return iid;
}

uint16_t calculate_ip_checksum(const uint16_t pairs[], size_t num_pairs)
{
    uint32_t sum = 0;
```

```
    for (size_t i = 0; i < num_pairs; i++) {

        if (i == 5) continue;

        sum += pairs[i];

        while (sum >> 16) {
            sum = (sum & 0xFFFF) + (sum >> 16);
        }
    }

    return ~((uint16_t)sum);
}

bool verify_ip_checksum(const uint16_t pairs[], size_t num_pairs) {
    uint32_t sum = 0;


    for (size_t i = 0; i < num_pairs; i++) {
        sum += pairs[i];


        while (sum >> 16) {
            sum = (sum & 0xFFFF) + (sum >> 16);
        }
    }

    return (sum & 0xFFFF) == 0xFFFF;
}


int classify_packet_priority(const char* packet) {
    if (strstr(packet, "URG") != NULL || strstr(packet, "HIGH") != NULL
|| strstr(packet, "PRIORITY=HIGH") != NULL) {
        return 1;
    } else if (strstr(packet, "ACK") != NULL || strstr(packet, "MEDIUM")
!= NULL || strstr(packet, "PRIORITY=MEDIUM") != NULL) {
        return 2;
    } else if (strstr(packet, "DATA") != NULL || strstr(packet, "LOW") !=
NULL || strstr(packet, "PRIORITY=LOW") != NULL) {
        return 3;
    } else {
        return 4;
    }
```

```
}


// --------- Funções de Diagnóstico de Rede ---------

void ping(const char* ip) {

    char command[100];

    #ifdef _WIN32
        snprintf(command, sizeof(command), "ping -n 4 %s", ip);
    #else
        snprintf(command, sizeof(command), "ping -c 4 %s", ip);
    #endif

    system(command);
}

void tracert(const char* ip) {

    char command[100];

    #ifdef _WIN32
        snprintf(command, sizeof(command), "tracert /h 5 %s", ip);
    #else
        snprintf(command, sizeof(command), "traceroute -m 5 %s", ip);
    #endif

    system(command);
}

// --------- Funções de Validação ---------

bool validate_ip(const char* ip) {
    int octet;
    char extra;

    if (sscanf(ip, "%d.%d.%d.%d%c", &octet, &octet, &octet, &octet,
&extra) == 4) {
        int parts[4];
        sscanf(ip, "%d.%d.%d.%d", &parts[0], &parts[1], &parts[2],
&parts[3]);
        for (int i = 0; i < 4; i++) {
            if (parts[i] < 0 || parts[i] > 255) return false;
        }
```

```
        return true;
    }
    return false;
}

bool validate_ipv6(const char* ipv6) {
    struct in6_addr result;
    return ipv4_pton(AF_INET6, ipv6, &result) == 1;
}

bool validate_cidr(int prefix_length) {
    return prefix_length >= 0 && prefix_length <= 128;
}

bool validate_mask(const char* mask) {
    int parts[4];

    if (sscanf(mask, "%d.%d.%d.%d", &parts[0], &parts[1], &parts[2],
&parts[3]) == 4) {
        int valid_masks[] = {255, 254, 252, 248, 240, 224, 192, 128, 0};
        for (int i = 0; i < 4; i++) {
            bool valid = false;
            for (int j = 0; j < 9; j++) {
                if (parts[i] == valid_masks[j]) {
                    valid = true;
                    break;
                }
            }
            if (!valid) return false;
        }
        return true;
    }
    return false;
}

bool validate_mac(const char* mac) {
    int values[6];
    char extra;

    if (sscanf(mac, "%2x:%2x:%2x:%2x:%2x:%2x%c",
               &values[0],    &values[1],    &values[2],    &values[3],
&values[4], &values[5], &extra) == 6) {
        return true;
    }
    return false;
```

```
}
```

7.3. APÊNDICE C

Código do arquivo de teste da biblioteca “main.c”:

```
#include <stdio.h>
#include <stdlib.h>
#include "network_utils.h"

//Para Linux
/*gcc -shared -o libnetworkutils.so network_utils.c -fPIC -lm -lgmp*/
/*gcc -o main main.c -L. -lnetworkutils -Wl,-rpath=. -lm -lgmp*/

//Para Windows
/*gcc -shared -o libnetworkutils.dll network_utils.c -fPIC -lm -lgmp -
lws2_32*/
/*gcc -o main main.c -L. -lnetworkutils -I. -lm -lgmp -lws2_32*/

int main() {

// Correção dos caracteres na execução do código
#ifdef _WIN32
        system("chcp 65001 > nul");
    #endif

    // Título
    printf("Exemplos de Funções da Biblioteca Network Utils:\n\n");

    // Conversões de base
printf("\n<-- Funções de Conversão IPv4, IPv6 e Resumo de IPv6 -->\n\n");

    int decimal = 10;
    char* binary = decimal_to_binary(decimal);
    if (binary) {
        printf("Decimal: %d\nEm binário: %s\n", decimal, binary);
    } else {
        fprintf(stderr, "Erro na conversão decimal para binário\n");
    }
    printf("\n");

    int decimal_from_binary = binary_to_decimal(binary);
    printf("Binário: %s\nEm decimal: %d\n", binary, decimal_from_binary);
printf("\n");
```

```
char*                       binary_from_hexadecimal                       =
hexadecimal_to_binary(decimal_to_hexadecimal(decimal));
    printf("Hexadecimal:          %s\nEm          binário:          %s\n",
decimal_to_hexadecimal(decimal), binary_from_hexadecimal);
    printf("\n");

    char* hexadecimal_from_binary = binary_to_hexadecimal(binary);
    printf("Binário:     %s\nEm     hexadecimal:     %s\n",     binary,
hexadecimal_from_binary);
    free(binary);
printf("\n");

    char* hex = decimal_to_hexadecimal(decimal);
    if (hex) {
        printf("Decimal: %d\nEm hexadecimal: %s\n", decimal, hex);
    } else {
        fprintf(stderr, "Erro na conversão decimal para hexadecimal\n");
    }
    printf("\n");

    int decimal_from_hex = hexadecimal_to_decimal(hex);
    printf("Hexadecimal: %s\nEm decimal: %d\n", hex, decimal_from_hex);
    free(hex);
printf("\n");

    // Conversões de IP para binário e vice-versa
    const char* ip = "192.168.1.100";
    char* binary_ip = ip_to_binary(ip);
    if (binary_ip) {
        printf("IP: %s\nEm binário: %s\n", ip, binary_ip);
    } else {
        fprintf(stderr, "Erro na conversão de IP para binário\n");
    }
    printf("\n");

    char* ip_from_binary = binary_to_ip(binary_ip);
    if (ip_from_binary) {
        printf("IP em binário: %s\nConvertido para IP: %s\n", binary_ip,
ip_from_binary);
        free(ip_from_binary);
       free(binary_ip);
    } else {
        fprintf(stderr, "Erro na conversão de binário para IP\n");
    }
    printf("\n");
```

```
    // Conversões de IPv6 para binário e vice-versa
    const char* ipv6 = "2001:0db8:85a3:0000:0000:8a2e:0370:7334";
    char* binary_ipv6 = ipv6_to_binary(ipv6);
    if (binary_ipv6) {
        printf("IPv6: %s\nEm binário: %s\n", ipv6, binary_ipv6);
    } else {
        fprintf(stderr, "Erro na conversão de IPv6 para binário\n");
    }
    printf("\n");

    char* ipv6_from_binary = binary_to_ipv6(binary_ipv6);
    if (ipv6_from_binary) {
        printf("IPv6 em binário: %s\nConvertido para IPv6: %s\n",
binary_ipv6, ipv6_from_binary);
        free(ipv6_from_binary);
       free(binary_ipv6);
    } else {
        fprintf(stderr, "Erro na conversão de binário para IPv6\n");
    }
    printf("\n");

    // Resumo de IPv6
    char* summarized_ipv6 = summarize_ipv6(ipv6);
    if (summarized_ipv6) {
printf("Endereço IPv6 original: %s\nEndereço IPv6 resumido: %s\n", ipv6,
summarized_ipv6);
free(summarized_ipv6);
} else {
    fprintf(stderr, "Erro na simplificação de IPv6\n");
}
printf("\n");


// Operações de máscaras de sub-rede
printf("\n<-- Operações de CIDR e Máscaras -->\n\n");

    int cidr = 24;
    char* mask_from_cidr = cidr_to_mask(cidr);
    printf("CIDR: %d\nMáscara: %s\n", cidr, mask_from_cidr);
    printf("\n");

int cidr_from_mask = mask_to_cidr(mask_from_cidr);
    printf("Máscara: %s\nCIDR: %d\n", mask_from_cidr, cidr_from_mask);
    printf("\n");
```

```
    char* network_ip = calculate_network_address(ip, cidr);
    if (network_ip) {
        printf("Endereço de rede: %s\n", network_ip);
    }
    printf("\n");

    char* broadcast_ip = calculate_broadcast_address(ip, cidr);
    if (broadcast_ip) {
        printf("Endereço de broadcast: %s\n", broadcast_ip);
    }
    printf("\n");

    char* first_host = calculate_first_host(network_ip);
    if (first_host) {
        printf("Primeiro Host: %s\n", first_host);
        free(first_host);
    } else {
        printf("Erro ao calcular o primeiro host.\n");
    }
    printf("\n");

    char* last_host = calculate_last_host(broadcast_ip);
    if (last_host) {
        printf("Último Host: %s\n", last_host);
       free(broadcast_ip);
        free(last_host);
    } else {
        printf("Erro ao calcular o último host.\n");
    }
printf("\n");

    int number_of_hosts = calculate_number_of_hosts(cidr);
    printf("Número de hosts em uma sub-rede /%d: %d\n", cidr,
number_of_hosts);
    printf("\n");

/*Equivalentes IPv6*/
printf("-> Funções Equivalentes para IPv6\n\n");

char* mask_ipv6 = cidr_to_mask_ipv6(cidr);
    printf("CIDR:      %d\nMáscara      (IPv6):      %s\n",      cidr,
cidr_to_mask_ipv6(cidr));
    printf("\n");
```

```
    printf("Máscara (IPv6): %s\nCIDR: %d\n", mask_ipv6, cidr);
    printf("\n");

char* network_ipv6 = calculate_network_address_ipv6(ipv6, cidr);
    if (network_ipv6) {
        printf("Endereço de rede (IPv6): %s\n", network_ipv6);
    }
    printf("\n");

    char* broadcast_ipv6 = calculate_broadcast_address_ipv6(ipv6, cidr);
    if (broadcast_ipv6) {
        printf("Endereço de broadcast (IPv6): %s\n", broadcast_ipv6);
    }
    printf("\n");

char* first_host_ipv6 = calculate_first_host_ipv6(network_ipv6);
    if (first_host_ipv6) {
        printf("Primeiro Host (IPv6): %s\n", first_host_ipv6);
        free(first_host_ipv6);
        free(network_ipv6);
} else {
        printf("Erro ao calcular o primeiro host.\n");
    }
    printf("\n");

char* last_host_ipv6 = calculate_last_host_ipv6(broadcast_ipv6);
    if (last_host_ipv6) {
        printf("Último Host (IPv6): %s\n", last_host_ipv6);
        free(last_host_ipv6);
       free(broadcast_ipv6);
} else {
        printf("Erro ao calcular o último host.\n");
    }
printf("\n");

char* number_of_hosts_ipv6 = calculate_number_of_hosts_ipv6(cidr);
    if (number_of_hosts_ipv6) {
        printf("Número de hosts em uma sub-rede /%d (IPv6): %s\n", cidr,
number_of_hosts_ipv6);
        free(number_of_hosts_ipv6);
} else {
        printf("Erro: CIDR inválido. Deve estar entre 0 e 128.\n");
    }
    printf("\n");
```

```
    // Super-rede
printf("\n<-- Funções de Super-Rede e Sub-Rede -->\n\n");
    printf("Para as redes 192.168.1.0 e 192.168.1.128, a Super-rede será:
");
    const char* networks[] = { "192.168.1.0", "192.168.1.128" };
    char* supernet = calculate_supernet(networks, 2);
    if (supernet) {
        printf("%s\n", supernet);
        free(supernet);
    }
    printf("\n");

    // Subnetting
    char* subnets = divide_subnets(network_ip, cidr, 4);
if (subnets) {
        printf("Sub-redes: %s", subnets);
        free(subnets);
       free(network_ip);
    }
    printf("\n");

    // Exemplo de Roteamento
printf("\n<-- Funções de Roteamento -->\n\n");

const char* first_destination = "192.168.1.0";
const char* first_mask = "255.255.255.0";
const char* first_gateway = "192.168.1.1";
    char*   first_route   =   add_route(first_destination,   first_mask,
first_gateway);
printf("%s", first_route);
const char* second_destination = "192.168.2.0";
const char* second_mask = "255.255.255.0";
const char* second_gateway = "192.168.2.1";
    char*  second_route  =  add_route(second_destination,  second_mask,
second_gateway);
printf("%s", second_route);
const char* third_destination = "10.0.0.0";
const char* third_mask = "255.0.0.0";
const char* third_gateway = "10.0.0.1";
    char*   third_route   =   add_route(third_destination,   third_mask,
third_gateway);
printf("%s", third_route);
printf("\n");
```

```
    char* routes_table = print_routes();
printf("%s", routes_table);
printf("\n");

    int match_index = longest_prefix_match(ip);
    if (match_index != -1) {
        printf("IP: %s\nMelhor correspondência de rota: %s\n", ip,
routing_table[match_index].destination);
} else {
        printf("Nenhuma correspondência de rota encontrada para o IP:
%s\n",ip);
    }
printf("\n");

    const char* ip_to_find = "192.168.1.0";
    const char* gateway = find_route(ip_to_find);
    if (gateway) {
        printf("A rota para %s tem o gateway: %s\n", ip_to_find, gateway);
    } else {
        printf("Nenhuma rota encontrada para o IP: %s\n", ip_to_find);
    }
printf("\n");

    char* remove_first_route = remove_route("192.168.1.0");
printf("%s", remove_first_route);
char* remove_second_route = remove_route("192.168.2.0");
printf("%s", remove_second_route);
char* remove_third_route = remove_route("10.0.0.0");
printf("%s", remove_third_route);
printf("\n");

routes_table = print_routes();
printf("%s", routes_table);
free(routes_table);
printf("\n");

//Exemplos de Análise de Tráfego
printf("\n<--  Funções de Análise de Tráfego -->\n\n");

    const char* mac = "AA:BB:CC:DD:EE:FF";
    char* iid = mac_to_iid(mac);
if (iid) {
        printf("Endereço MAC: %s\nInterface Identifier (IID) gerado a
partir do MAC: %s\n", mac, iid);
        free(iid);
```

```
} else {
        printf("Erro ao alocar memória para geração de IID.\n");
    }
printf("\n");

uint16_t ip_header[] = {0x4500, 0x0073, 0x0000, 0x4000, 0x4011, 0xb861,
0xc0a8, 0x0001, 0xc0a8, 0x00c7};
    uint16_t      checksum      =      calculate_ip_checksum(ip_header,
ARRAY_SIZE(ip_header));
printf("Cabeçalho IP: ");
    for (size_t i = 0; i < ARRAY_SIZE(ip_header); i++) {
        printf("0x%04X ", ip_header[i]);
    }
    printf("\n");
printf("Checksum calculado: 0x%04X\n", checksum);
    printf("\n");

bool is_valid = verify_ip_checksum(ip_header, ARRAY_SIZE(ip_header));
printf("O checksum é válido? %s\n", is_valid ? "Sim." : "Não.");
printf("\n");

const char* packet_high_priority = "URG DATA PRIORITY=HIGH";
    const char* packet_medium_priority = "ACK PRIORITY=MEDIUM";
    const char* packet_low_priority = "DATA PRIORITY=LOW";
    const char* packet_undefined_priority = "NO_PRIORITY";
    printf("Classificação de Prioridade dos Pacotes:\n");
    printf("Pacote  1:  '%s'\nPrioridade:  %d\n",  packet_high_priority,
classify_packet_priority(packet_high_priority));
    printf("Pacote  2:  '%s'\nPrioridade:  %d\n",  packet_medium_priority,
classify_packet_priority(packet_medium_priority));
    printf("Pacote  3:  '%s'\nPrioridade:  %d\n",  packet_low_priority,
classify_packet_priority(packet_low_priority));
    printf("Pacote 4: '%s'\nPrioridade: %d\n", packet_undefined_priority,
classify_packet_priority(packet_undefined_priority));
    printf("\n");

    // Diagnóstico de Rede
printf("\n<-- Funções de Diagnóstico de Rede -->\n\n");
const char* test_ip = "172.20.235.153";
    ping(test_ip);
    printf("\n");

    tracert(test_ip);
    printf("\n");
```

```
    // Validação
printf("\n<-- Funções de Validação -->\n\n");

if (validate_ip(ip)) {
        printf("Endereço IP: %s\nVálido.\n", ip);
    } else {
        printf("Endereço IP: %s\nInválido.\n", ip);
    }
printf("\n");

const char* invalid_ip = "192.1683.0.1";
if (validate_ip(invalid_ip)) {
        printf("Endereço IP: %s\nVálido.\n", invalid_ip);
    } else {
        printf("Endereço IP: %s\nInválido.\n", invalid_ip);
    }
printf("\n");

if (validate_ipv6(ipv6)) {
        printf("Endereço IPv6: %s\nVálido.\n", ipv6);
    } else {
        printf("Endereço IPv6: %s\nInválido.\n", ipv6);
    }
printf("\n");

const char* invalid_ipv6 = "::0db8:85a3:0000:0000:8a2e:0370::";
    if (validate_ipv6(invalid_ipv6)) {
        printf("Endereço IPv6: %s\nVálido.\n", invalid_ipv6);
    } else {
        printf("Endereço IPv6: %s\nInválido.\n", invalid_ipv6);
    }
printf("\n");

if (validate_cidr(cidr)) {
        printf("CIDR: /%d\nVálido.\n", cidr);
    } else {
        printf("CIDR: /%d\nInválido.\n", cidr);
    }
printf("\n");

    int invalid_cidr = 156;
if (validate_cidr(invalid_cidr)) {
        printf("CIDR: /%d\nVálido.\n", invalid_cidr);
    } else {
        printf("CIDR: /%d\nInválido.\n", invalid_cidr);
```

```
    }
printf("\n");

if (validate_mask(mask_from_cidr)) {
        printf("Máscara: %s\nVálida.\n", mask_from_cidr);
    } else {
        printf("Máscara: %s\nInválida.\n", mask_from_cidr);
    }
free(mask_from_cidr);
printf("\n");

    const char* invalid_mask = "255..255.0";
if (validate_mask(invalid_mask)) {
        printf("Máscara: %s\nVálida.\n", invalid_mask);
    } else {
        printf("Máscara: %s\nInválida.\n", invalid_mask);
    }
printf("\n");

if (validate_mac(mac)) {
        printf("Endereço MAC: %s\nVálido.\n", mac);
    } else {
        printf("Endereço MAC: %s\nInválido.\n", mac);
    }
printf("\n");

const char* invalid_mac = "AA:BB:CC:DD:GG:FF";
    if (validate_mac(invalid_mac)) {
        printf("Endereço MAC: %s\nVálido.\n", invalid_mac);
    } else {
        printf("Endereço MAC: %s\nInválido.\n", invalid_mac);
    }
printf("\n");

printf("\nPressione qualquer tecla para continuar...");
    getchar(); // Espera uma tecla ser pressionada


    return 0;
}
```