

Trabalho de Programação de Redes

Anderson Madeira¹, Gabriel Velasco¹, Marco Ticona¹

¹Fundação Universidade Federal do Pampa (UNIPAMPA)

Abstract. *In this article, the implementation of the first version of the HTTP server project is presented, in which HTTP 1.0 developed in the C programming language is implemented. Furthermore, the results of the tests related to this version are presented and the results are discussed.*

Resumo. *Neste artigo é apresentada a implementação da primeira versão do projeto do servidor HTTP, em que é implementado o HTTP 1.0 desenvolvido na linguagem em programação C. Além disso, é apresentado os resultados dos testes em relação a essa versão e discutido os resultados.*

1. Introdução

Um servidor HTTP (*HyperText Transfer Protocol*) possui a atribuição de prestar serviços de armazenamento, processamento e entrega de páginas web aos clientes solicitantes. Em que é utilizado o protocolo HTTP como forma de comunicação entre um servidor e um cliente [Docs 2021].

De forma ainda mais completa, HTTP é um protocolo da camada de aplicação para a transmissão de documentos de hipermídia, como o HTML (*HyperText Markup Language*). Este foi desenvolvido para comunicação entre navegadores web e servidores web, porém pode ser utilizado para outros propósitos. Este protocolo modelo cliente-servidor clássico, em que um cliente abre uma conexão, executa uma requisição e espera até receber uma respostas. Este protocolo tem a característica de ser sem estado, ou seja, o servidor não mantém nenhum dado entre duas requisições. Apesar de ser frequentemente baseado em uma camada que implementa o modelo TCP (*Transmission Control Protocol*)/IP (*Internet Protocol*) [Forouzan 2013]. Pode ser utilizado em qualquer camada de transporte confiável, ou seja, um protocolo que não perde mensagens silenciosamente como o UDP (*User Datagram Protocol*) [Rodrigues 2021].

Nesta trabalho, é apresentada a primeira versão do projeto de redes, em que é implementada a versão HTTP 1.0 para diversos clientes e sem conexões persistentes.

2. Objetivos

- Implementar o protocolo HTTP 1.0;
- Implementar conexões não persistentes suportando diversos clientes.

3. Metodologia

Inicialmente foi realizado um estudo utilizando os materiais sobre *sockets* [Forouzan 2013] e sobre o protocolo HTTP baseado nos conceitos apresentados por [Docs 2021], [Rodrigues 2021] e [Forouzan 2013], nas aulas de rede computadores e nas vídeo-aulas sobre a protocolos da camada de aplicação da Universidade Virtual do Estado de São Paulo (UNIVESP).

Foram utilizadas ferramentas como o *GitHUB* para versionamento do código, bem como o aplicativo de comunicação *Discord* para se realizar as reuniões referentes ao trabalho. O servidor em si, foi desenvolvido na linguagem de programação C, na *IDE Visual Code Studio* no sistema operacional *Windows*, porém os testes foram realizados utilizando uma máquina virtual do *Ubuntu 20.04* e a ferramenta de extensão WSL para compilar as aplicações C em *Linux*.

Primeiramente, são definidos as *socket* (porta) de comunicação entre o servidor e o cliente e o caminho onde se encontra o arquivo. O servidor então aplica o *parser* nas linhas de comando em busca de erros e de saber quais comandos foram passados nos argumentos. O servidor então apresenta uma mensagem indicando em qual porta a comunicação se iniciou e em qual diretório serão realizadas as ações do cliente.

Após os primeiros passos, é então chamada uma função *starServer()* para se iniciar a comunicação entre o servidor e o cliente, em que a função obtém os endereçamento, abre o *socket()* para se "ouvido" e utiliza o *bind()* para associar o endereço a ele. Ainda na função *starServer()* existem verificações para erros quanto a utilização do *socket()* e para a escuta de conexões que ainda serão geradas por novas solicitações dos usuários.

É configurado um vetor de clientes, em que cada espaço desse vetor é interpretado como um *slot* ocupado ou que será ocupado por um cliente. Sendo utilizado o *accept* para aceitar novas conexões e atribui as mesmas a um *slot* do vetor clientes. São utilizadas *pthread* para implementar múltiplos clientes fazendo requisições simultâneas, porém não mantendo a conexão persistente, ou seja, o cliente não faz novas solicitações após já ter realizado uma.

Por fim, uma função *respond()* foi criada para emular as respostas do servidor ao cliente, em que essa função recebe informações do cliente, do endereçamento e de outras informações complementares. Em que são verificadas as informações recebidas para saber se existe erro ou para saber se a conexão foi encerrada. Caso não exista problemas quanto as informações passadas, é então implementado o método *GET*. Há também o tratamento do endereço passado, em que caso o usuário apenas apresente um endereço do tipo *Localhost:8080* em algum *browser*, o servidor irá adicionar por padrão um "*index.html*" para prover um tipo de arquivo a ser aberto. Porém o servidor pode lidar com diversos tipos de arquivos como .txt, .html, .pdf, .png, etc. Por fim, o servidor encontra o arquivo a ser utilizado e retorna mensagens quanto as verificações de que se todos os status estão "ok". Ao final, é finalizado a conexão fechando a porta com um *shutdown* e um *close*. Deve-se ressaltar que caso o arquivo especificado não esteja no diretório atual, deve-se prover o caminho completo para o qual o arquivo se encontra.

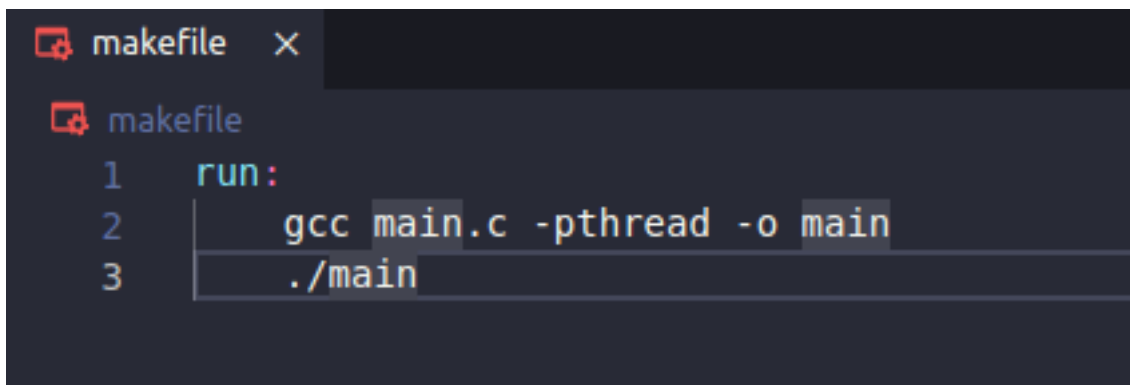
Para a execução do servidor no terminal é utilizada uma estrutura do tipo:

- Localhost:[porta]/[arquivo].[extensão do arquivo].

O programa foi desenvolvido todo em um único arquivo para prover simplificação, além de quem foi construído um arquivo *makefile* descrito na figura 1 para facilitar a compilação do programa. O programa completo pode ser acessado em: GITHUB.

4. Resultados e Discussões

Foi implementado uma versão que contém lida com diversas requisições de clientes, do tipo de conexão não persistente. Em que é possível prover serviços de arquivos do tipo



```
makefile x
makefile
1  run:
2    gcc main.c -pthread -o main
3    ./main
```

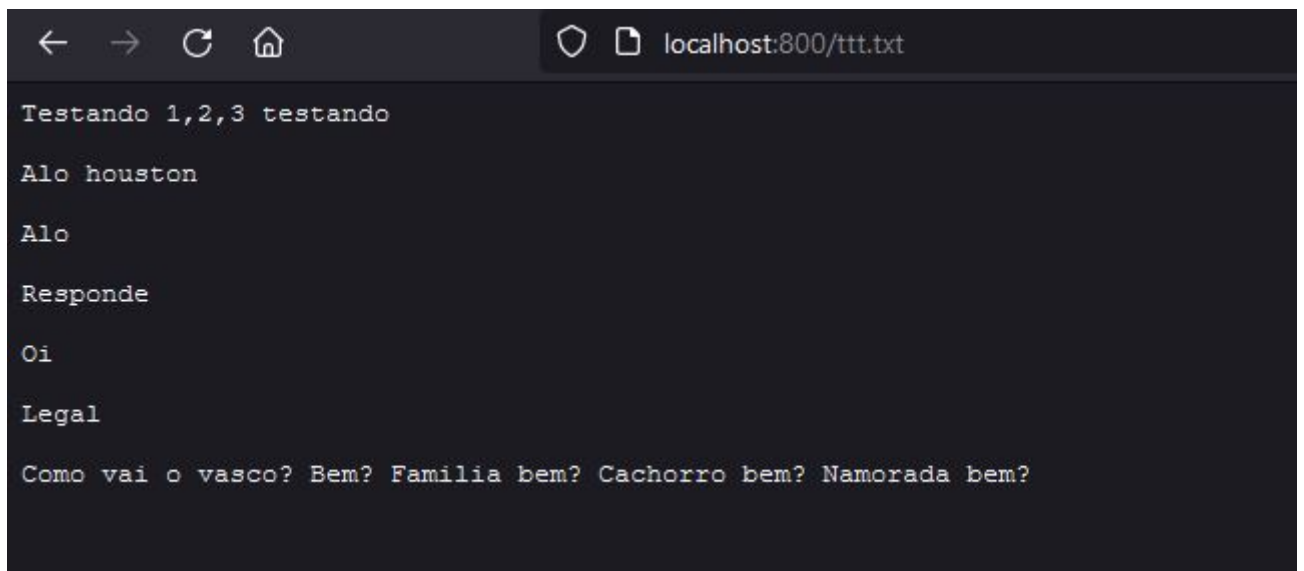
Figura 1. makefile

.txt, .pdf, .html, .png, dentre outros. A uma taxa definida de 1024 bytes por segundo.

Foi implementado também o mecanismo de *threads* utilizando *pthread* em que cada cliente é uma *thread* e podem realizar requisições simultâneas. Ainda deve-se realizar a implementação de um arquivo que contenha os IPs dos clientes com taxas de envio variada.

Na figura 2 é demonstrado um exemplo de funcionamento da aplicação, em que é requisitado o arquivo e provido um caminho:

- Arquivo requisitado: "ttd.txt".
- Caminho provido: "/mnt/c/Users/pirat/Desktop/ttd.txt"



```
localhost:800/ttd.txt
Testando 1,2,3 testando
Alo houston
Alo
Responde
Oi
Legal
Como vai o vasco? Bem? Familia bem? Cachorro bem? Namorada bem?
```

Figura 2. Arquivo teste

Dessa forma, pelo terminal é possível se obter diversas informações quanto ao tipo do protocolo, de qual fonte vem o arquivo, que tipo de arquivo está se lidando, dentre outras informações. Todas essas informações são apresentadas na figura 3.

Figura 3. Terminal do servido

```
Servidor iniciado na porta de numero. 800 com o caminho/diretorio provido sendo /mnt/c/Users/pirat/Desktop
GET /ttt.txt HTTP/1.1
Host: localhost:800
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1

file: /mnt/c/Users/pirat/Desktop/ttt.txt
HTTP/1.0 200 OK
```

Autor (2021)

5. Conclusão

Dessa forma, foi implementado com sucesso um servidor que recebe requisições concorrentes (utilizando *pthreads*) do tipo conexão não persistente utilizando o protocolo HTTP 1.0 como foi o objetivo proposto inicialmente. Nas próximas versos pretende-se implementar o arquivo que contem IPs com taxa de envio variáveis, além do protocolo HTTP 1.1 e conexões persistentes.

Referências

- [Docs 2021] Docs, M. W. (2021). Http tutoriais. In <https://developer.mozilla.org/pt-BR/docs/Web/HTTP>. Firefox.
- [Forouzan 2013] Forouzan, A. B. (2013). *Redes de computadores uma abordagem top-down*. AMGH, 1 edition.
- [Rodrigues 2021] Rodrigues, A. (2021). Como funciona um servidor http/web. In <https://www.portalgsti.com.br/2017/08/como-funciona-um-servidor-http-web.html>. Portal GSTI.