

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 second version of the HTTP server project is presented, in which HTTP 1.1 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 segunda versão do projeto do servidor HTTP, em que é implementado o HTTP 1.1 desenvolvido na linguagem de 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 [Kamienski 2021]. 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 segunda versão do projeto de redes, em que é implementada a versão HTTP 1.1 para diversos clientes e conexões persistentes.

2. Objetivos

- Implementar o protocolo HTTP 1.1;
- Implementar conexões persistentes suportando diversos clientes;
- Suportar conexões de outras máquinas usando o IP;
- Suportar páginas html com diversos conteúdos.

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], [Forouzan 2013] e [Everything], 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*. Os testes foram realizados utilizando uma máquina virtual do *Ubuntu 20.04*, uma máquina virtual *Ubuntu Server 20.40* e a ferramenta de extensão WSL para compilar as aplicações C em *Linux* e também no sistema operacional *Linux Mint*, usando o *browser Google Chrome*. Para o uso das máquinas virtuais foi utilizado a *Oracle VM VirtualBox v6.1*.

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:800* 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.

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

Figura 1. makefile

4. Resultados e Discussões

Foi implementado uma versão que lida com diversas requisições de clientes, do tipo de conexão persistente. Em que é possível prover serviços de arquivos do tipo .txt, .pdf, .html, .png, dentre outros. A uma taxa definida de 1024 bytes por segundo, a qual é apresentada toda vez que uma requisição é concluída.

Foi implementado também o mecanismo de *threads* utilizando *pthread* em que cada cliente é uma *thread* e podem realizar requisições simultâneas. Deve-se ressaltar que para que o servidor seja executado com sucesso é necessário executar o mesmo no modo de superusuário do Windows ou do Linux.

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

- Arquivo requisitado: "example.html".
- Estrutura da requisição [IP]:[PORTA];
- Caminho provido: "http://localhost:800/example.html".

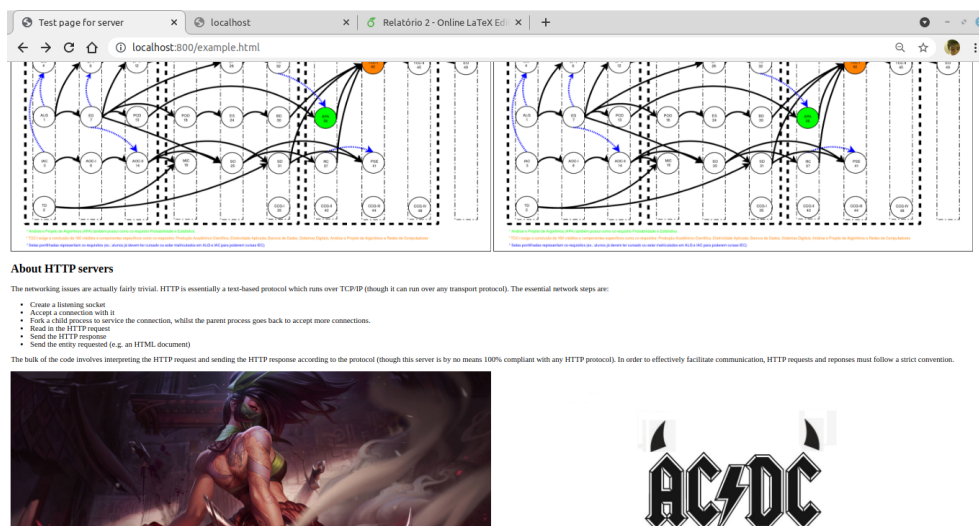


Figura 2. Arquivo teste de uma página contendo diversas requisições

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 servidor que contém as requisições

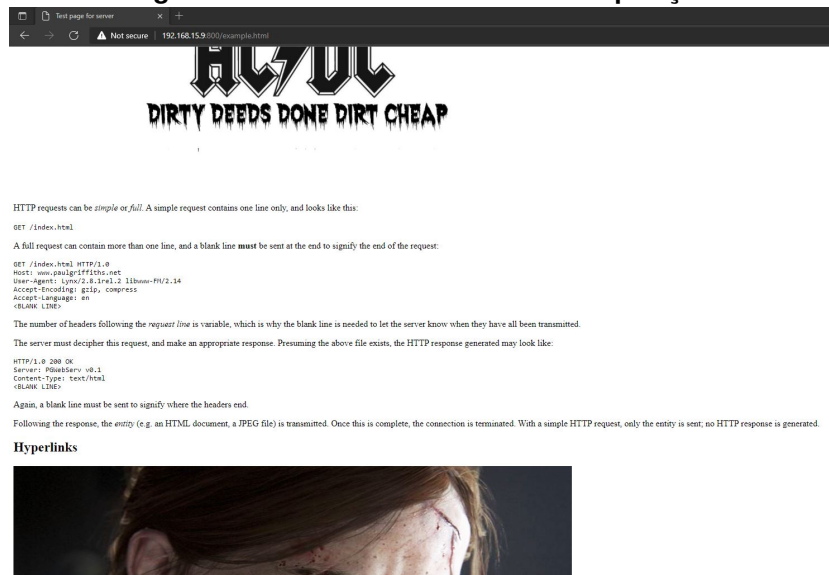
```
Referer: http://localhost:800/index.html
Accept-Encoding: gzip, deflate, br
Accept-Language: pt,en;q=0.9,en-US;q=0.8,es;q=0.7
Cookie: connect.sid=s%3AW9RSdEcioNuI7mMioHBNFFyToh3_a8vh.7ekv9xIBFa%2FYJosLW12dvbW2Y4D5AFm6ho060RV3Juc

file: /home/marco/Documents/Unipampa/9-Semestre/Redes/Projetos/Atividade/300.gif
GET /200.webp HTTP/1.1
Host: localhost:800
Connection: keep-alive
sec-ch-ua: "Chromium";v="92", " Not A;Brand";v="99", "Google Chrome";v="92"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:800/index.html
Accept-Encoding: gzip, deflate, br
Accept-Language: pt,en;q=0.9,en-US;q=0.8,es;q=0.7
Cookie: connect.sid=s%3AW9RSdEcioNuI7mMioHBNFFyToh3_a8vh.7ekv9xIBFa%2FYJosLW12dvbW2Y4D5AFm6ho060RV3Juc

file: /home/marco/Documents/Unipampa/9-Semestre/Redes/Projetos/Atividade/200.webp
Conexão encerrada, socket desconectado
Conexão encerrada, socket desconectado
GET /ellie.jpg HTTP/1.1
Host: localhost:800
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; 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
```

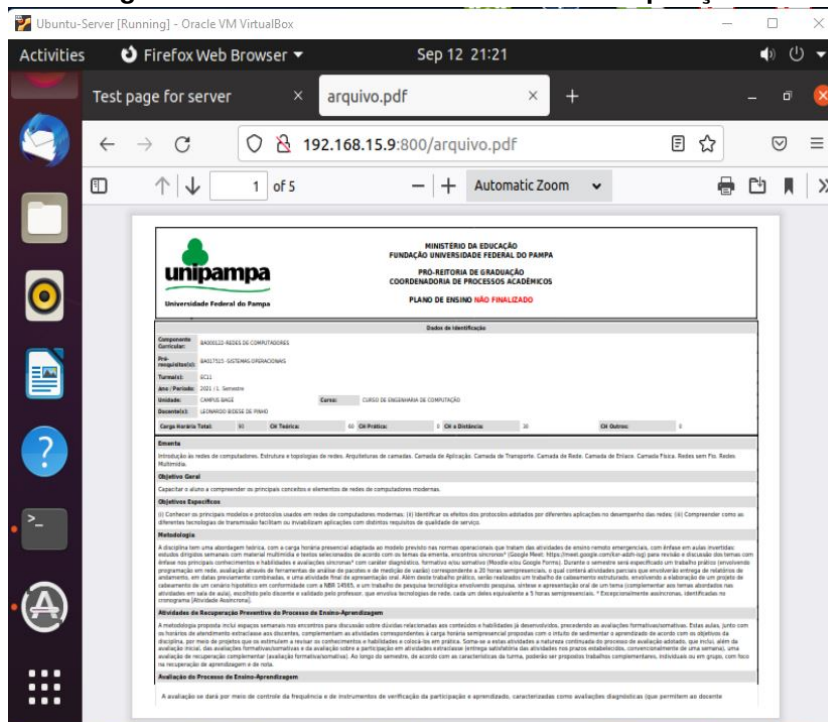
Foram performados testes usando máquinas diferentes que pediam requisições ao servidor utilizando a máquina que o mesmo está usando de host. Na figura 4 podemos observar o *browser* Microsoft Edge fazendo requisição ao servidor e recebendo uma página html completa com diversas figuras e texto. Passando no *browser* informações como IP (192.168.15.9) e a porta (800) que o servidor está ”escutando”.

Figura 4. Cliente Microsoft fazendo requisições



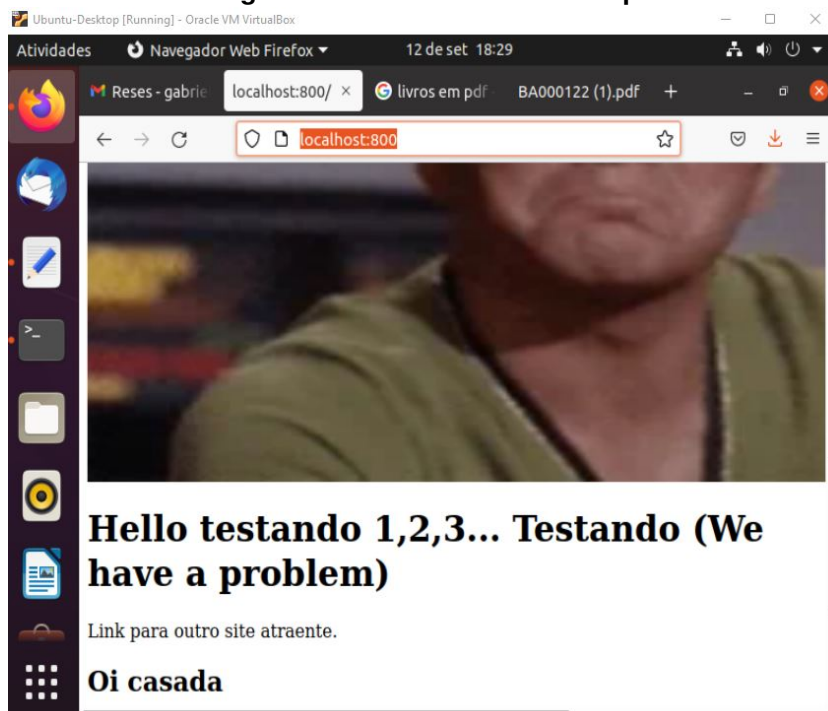
Na figura 5 vemos um cliente fazendo requisições de um .pdf através do *browser* Firefox em uma máquina contendo o Ubuntu-Server. O qual assim como no exemplo anterior, passa os mesmo parâmetros de IP e porta. Com isso o cliente consegue acessar de uma outra máquina a máquina que contém o servidor usando o IP da máquina host e adquirindo o objeto requisitado.

Figura 5. Cliente Ubuntu-Server fazendo requisições



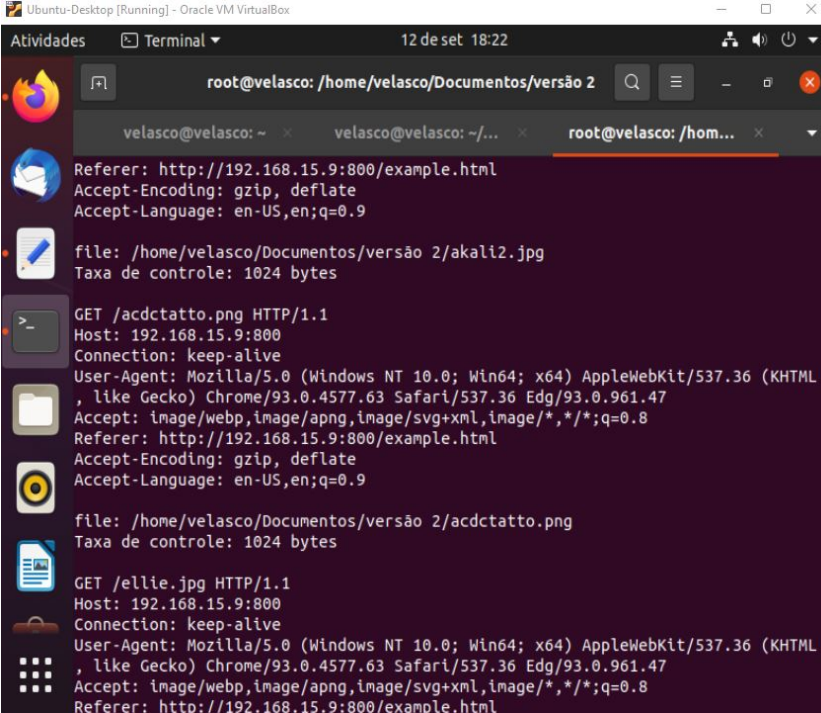
Já na figura 6 observamos que um cliente Ubuntu-Desktop fazendo uma requisição de uma página html completa contendo texto e imagens animadas, através do *browser* Firefox. O qual passa parâmetros para a requisição usando "localhost:800", o equivalente a digitar o próprio IP da máquina, visto que o servidor está contido na mesma.

Figura 6. Cliente Ubuntu-Desktop



Por fim, na figura 7 observamos que a mesma contém informações acerca das requisições feitas por diversos clientes, taxa de controle, endereço do host, informações do arquivo e seu caminho, o método implementado, o cliente que fez a solicitação (ex: Firefox, Microsoft Edge, etc), dentre outras diversas informações.

Figura 7. Informações acerca da execução do terminal



```
root@velasco: /home/velasco/Documentos/versão 2
Referer: http://192.168.15.9:800/example.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

file: /home/velasco/Documentos/versão 2/akali2.jpg
Taxa de controle: 1024 bytes

GET /acdctatto.png HTTP/1.1
Host: 192.168.15.9:800
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.63 Safari/537.36 Edg/93.0.961.47
Accept: image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.15.9:800/example.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

file: /home/velasco/Documentos/versão 2/acdctatto.png
Taxa de controle: 1024 bytes

GET /ellie.jpg HTTP/1.1
Host: 192.168.15.9:800
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.63 Safari/537.36 Edg/93.0.961.47
Accept: image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.15.9:800/example.html
```

5. Conclusão

Dessa forma, foi implementado com sucesso um servidor que recebe requisições concorrentes (utilizando *pthreads*) do tipo conexão persistente utilizando o protocolo HTTP 1.1 como foi o objetivo proposto inicialmente, além de ter uma taxa controlada de 1024 bytes por segundo e múltiplos acessos tanto na mesma máquina (usando o localhost) quanto recebendo requisições de diferentes máquinas ([IP da máquina host]:[PORTA]), em que essas requisições podem ser páginas completas de html, arquivos de imagens, imagens animadas, .pdf, dentre outros. Não foi possível implementar o arquivo que contém IPs com taxas de envio variáveis, o processo de *pipelined* e implementar as diversas questões de QoS (*Quality of Service*).

Referências

- [Docs 2021] Docs, M. W. (2021). Http tutoriais. In <https://developer.mozilla.org/pt-BR/docs/Web/HTTP>. Firefox.
- [Everything] Everything, S. Http server: Everything you need to know to build a simple http server from scratch.
- [Forouzan 2013] Forouzan, A. B. (2013). *Redes de computadores uma abordagem top-down*. AMGH, 1 edition.

[Kamienski 2021] Kamienski, C. (2021). Como funciona um servidor http/web. CIn/UFPE.

[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.