

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Atividade extra 01

Brasília, DF
2015



Lucas Albuquerque Medeiros de Moura 11.0015568

Thiago de Souza Fonseca Ribeiro 10.0125271

Luciano Prestes Cavalcanti 11.0035208

Atividade extra 01

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Tiago Alves

Brasília, DF

2015

1 Introdução

1.1 Introdução

Este documento trata dos aspectos técnicos e algumas especificades das soluções desenvolvidas para a primeira atividade extra da disciplina de fundamentos de redes de computadores.

A atividade extra consiste no desenvolvimento de duas aplicações. A primeira dela é um simples webserver capaz de tratar requisições *HTTP* do tipo *GET*. O webserver deve enviar requisições com código 404 caso o arquivo sendo requisitado não for encontrado na base de dados.

A segunda tarefa consiste na construção de um client ping capaz de calcular o *Round Trip Time* de um dado servidor.

2 Cliente Ping

2.1 Linguagem de programação

A linguagem de programação adotada foi o C++.

2.2 Sistema operacional

O sistema operacional usada para implementar a solução proposta foi a distro linux Debian.

2.3 Ambiente de desenvolvimento

Para o desenvolvimento da aplicação foram usados: um editor de texto e um compilador.

O editor de texto escolhido foi o vim e o compilador usado foi o g++.

Entretanto, para a execução dos testes desenvolvidos, é necessário não só o compilador citado no parágrafo anterior, mas também de algumas bibliotecas extras que devem ser manualmente instaladas, sendo elas a libcppunit-dev e libcppunit-doc.

O grupo utilizou git para o desenvolvimento do trabalho, visto que precisávamos trabalhar de forma destrubuída, dividindo tarefas para cada integrante do grupo, para que assim todos pudessem participar do trabalho.

O repositório git encontra-se em: [Repositório do projeto](#)

2.4 Implementação

A aplicação funciona basicamente por meio de três classes, HTTPParser, Socket e Worker.

A classe Socket é responsável pela comunicação entre o cliente e o servidor. Sendo por meio dela que a aplicação consegue receber requisições e enviar respostas. Vale ressaltar que o protocolo TCP foi usada na configuração do socket usado na aplicação.

Existem ainda 2 classes filhas chamadas ServerSocket e ClientSocket, a classe ClientSocket é a responsável por criar uma conexão com o socket em questão e também para fechar a conexão, também é dela o papel de gerenciar a troca de dados com recv() e send(). Já a classe ServerSocket é a responsável por atribuir um endereço ao Socket (bind),

isso é necessário para que ele possa receber conexões. Por fim temos implementado na classe `ServerSocket` o `acceptclient` que vai extrair a primeira conexão da fila de conexões pendentes para cada socket de escuta.

A classe `HTTPParser` é responsável por receber a requisição HTTP e separar a mesma em linha de requisição, cabeçalho e corpo da mensagem. Além disso, tal classe também é responsável por checar se linha de requisição é válida ou não.

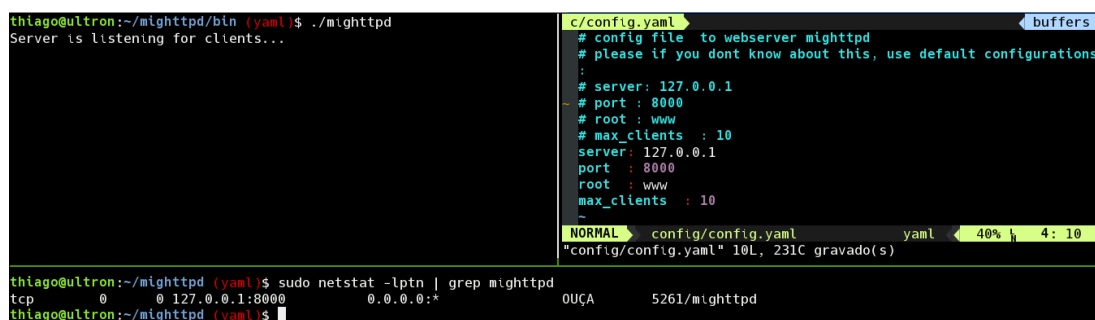
A classe `Worker` é responsável por administrar as duas classes citadas acima para prover uma resposta ao cliente. Vale ressaltar que cada vez que uma requisição HTTP é recebida, uma instância da classe `worker` é criada em nova thread é criada para tratar tal requisição. Sendo assim, o que essa classe basicamente faz e receber a requisição, extrair a primeira linha da mesma, checar se a requisição é válida e retornar o objeto requisitado. Vale ressaltar que se a aplicação não encontrar o arquivo sendo requisitado, a mesma irá retornar uma resposta com o código 404. Todos os arquivos que podem ser requisitados se encontram no diretório `www`.

2.5 Instruções de uso

Para compilar a aplicação, é necessário: possuir o compilador `g++`, estar no diretório raiz da aplicação e executar o comando `make`.

Uma vez compilada, deve-se ir ao diretório `bin` e executar o binário gerado (com o comando `./bin`). Para dizer ao servidor qual a porta em que ele deve executar e outras configurações importantes podem ser feitas no arquivo que se encontra em `config/config.yaml`. Existe um configuração default que pode ser seguida, a porta default é a porta 8000 e o server roda em 127.0.0.1 ou seja e localhost.

Nosso teste para ver se a aplicação estava escutando na porta certa foi utilizar o comando `netstat -lptn` que mostra as aplicações rodando e as portas em que elas estão escutando, a nossa aplicação está escutando na porta que foi configurada no arquivo `config.yaml` como podemos ver na figura abaixo:



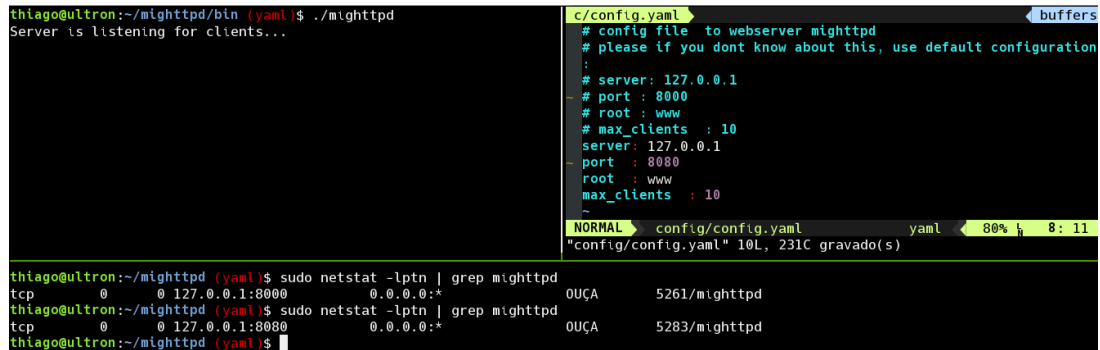
```
thiago@ultron:~/mighttpd/bin (yaml)$ ./mighttpd
Server is listening for clients...

c/config.yaml
# config file to webserver mighttpd
# please if you dont know about this, use default configurations
:
# server: 127.0.0.1
# port : 8000
# root : www
# max_clients : 10
server: 127.0.0.1
port : 8000
root : www
max_clients : 10
~
NORMAL config/config.yaml yaml 40% 4: 10
"config/config.yaml" 10L, 231C gravado(s)

thiago@ultron:~/mighttpd (yaml)$ sudo netstat -lptn | grep mighttpd
tcp        0      0 127.0.0.1:8000        0.0.0.0:*
thiago@ultron:~/mighttpd (yaml)$
```

Figura 1: Servidor escutando na porta 8000

Também é possível trocar a configuração para que façam testes em outra porta, trocamos a porta no config.yaml para 8080 e realizamos o mesmo teste como na figura abaixo:



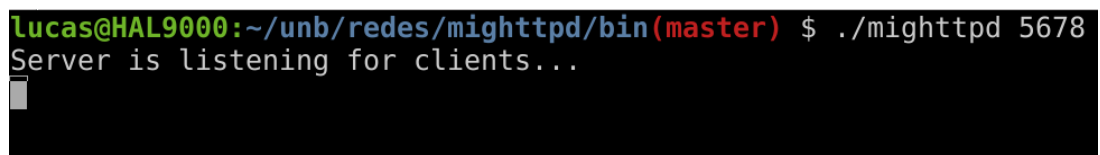
```
thiago@ultron:~/mighttpd/bin (yaml)$ ./mighttpd
Server is listening for clients...

c/config.yaml
# config file to webserver mighttpd
# please if you dont know about this, use default configuration
:
# server: 127.0.0.1
# port : 8000
# root : www
# max_clients : 10
server: 127.0.0.1
port : 8080
root : www
max_clients : 10
~
NORMAL config/config.yaml yaml 80% 8: 11
"config/config.yaml" 10L, 231C gravado(s)

thiago@ultron:~/mighttpd (yaml)$ sudo netstat -ltn | grep mighttpd
tcp        0      0 127.0.0.1:8000        0.0.0.0:*           0UÇA      5261/mighttpd
thiago@ultron:~/mighttpd (yaml)$ sudo netstat -ltn | grep mighttpd
tcp        0      0 127.0.0.1:8080        0.0.0.0:*           0UÇA      5283/mighttpd
thiago@ultron:~/mighttpd (yaml)$
```

Figura 2: Servidor escutando agora em outra porta

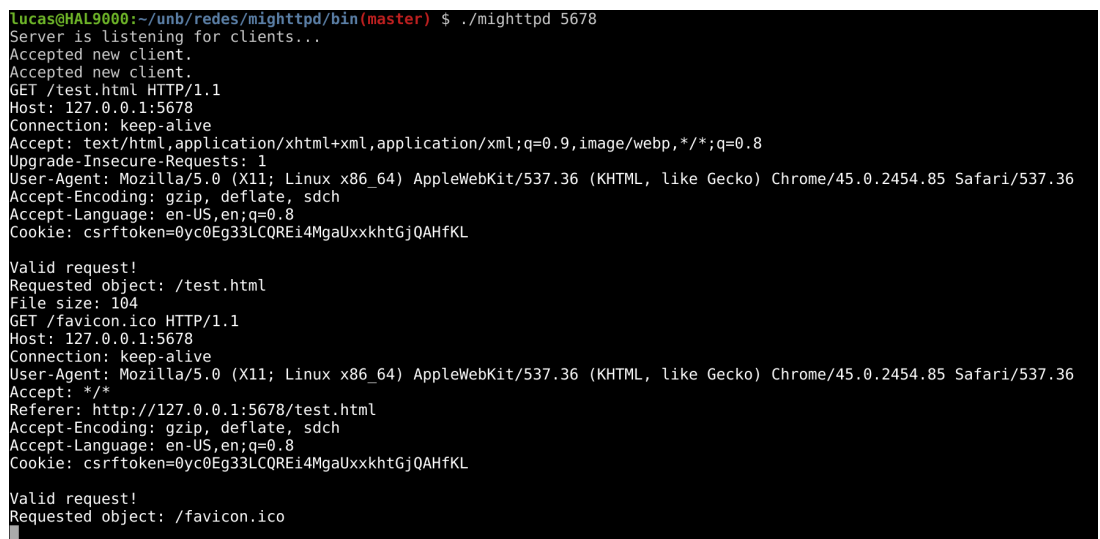
Com a aplicação já em funcionamento, a mesma irá esperar por uma requisição de algum cliente, pode ser visto na figura à seguir:



```
Lucas@HAL9000:~/unb/redes/mighttpd/bin(master) $ ./mighttpd 5678
Server is listening for clients...
```

Figura 3: Aplicação esperado por requisição

Uma vez que o cliente mande uma requisição, a aplicação irá mostrar a requisição feita e então irá gerar uma resposta apropriada e enviar ao cliente.



```
Lucas@HAL9000:~/unb/redes/mighttpd/bin(master) $ ./mighttpd 5678
Server is listening for clients...
Accepted new client.
Accepted new client.
GET /test.html HTTP/1.1
Host: 127.0.0.1:5678
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: csrftoken=0yc0Eg33LCQREi4MgaUxxkhtGjQAHfKL

Valid request!
Requested object: /test.html
File size: 104
GET /favicon.ico HTTP/1.1
Host: 127.0.0.1:5678
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.85 Safari/537.36
Accept: */*
Referer: http://127.0.0.1:5678/test.html
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cookie: csrftoken=0yc0Eg33LCQREi4MgaUxxkhtGjQAHfKL

Valid request!
Requested object: /favicon.ico
```

Figura 4: Servidor recebendo uma requisição

2.6 Testes

A aplicação foi testada de duas formas, testes manuais e unitários.

Os testes manuais foram executados por meio do uso de browsers por checar se a aplicação estava respondendo de forma apropriada as requisições sendo feitas.

Os testes unitários foram usados para testar o comportamento de algumas funções cruciais para a execução da aplicação, como alguns métodos da classe HTTPParser. Para executar os testes unitários, basta executar o comando `make run_tests` na raiz da aplicação. Vale ressaltar que as dependências citadas na seção Ambiente de desenvolvimento deve estar previamente instaladas.

2.7 Limitações

Atualmente a aplicação funciona apenas com para requisições GET. Outro fato que precisa ser mencionado é o fato de que a aplicação consegue quebrar a requisição feita em em blocos distintos, mas não está usando nenhuma informação além da própria linha de requisição, ou seja, a aplicação não está tratando cabeçalhos e o corpo da mensagem da aplicação.

Outro ponto que merece destaque é que a aplicação também só consegue enviar resposta para os casos do objeto sendo requisitado ter sido encontrado e não encontrado, ou seja, a aplicação não consegue fornecer resposta apropriadas para algumas situações, como versão o HTML inválida, dentre outros casos.

Por fim, a aplicação foi testada apenas com os browser chrome e iceweasel, dessa forma, o grupo não consegue afirmar com certeza que a aplicação irá funcionar em outros browser não testados, como internet explorer. Entretanto, vale ressaltar que o grupo não conseguiu identificar nenhum problema que talvez leve a aplicação a não funcionar em outros browsers.

Referências