Programação Web
2019 / 2020
MEIT

Prática: Servidor de HTTP/1.0

UNIVERSIDADE AUTÓNOMA DE LISBOA

Introdução

Nesta prática iremos desenvolver em Python um pequeno servidor que implemente parcialmente o protocolo HTTP/1.0, usando *sockets* TCP/IP. Esta prática tem por base a implementação inicial do servidor (código *httpserver.py*) e dois ficheiros *html* dentro da pasta *htdocs*, tudo no repositório fornecido.

Coloque os ficheiros na sua máquina. Crie um projecto no seu IDE com estes ficheiros, e execute o código do servidor (*httpserver.py*). O servidor estará à escuta no porto 8000. Execute o *browser* do seu computador e abra a página "http://localhost:8000/". Se tudo estiver correto, deverá ver o "Olá Mundo!" no navegador, possivelmente com problemas nos caracteres da nossa língua (como corrigir?)

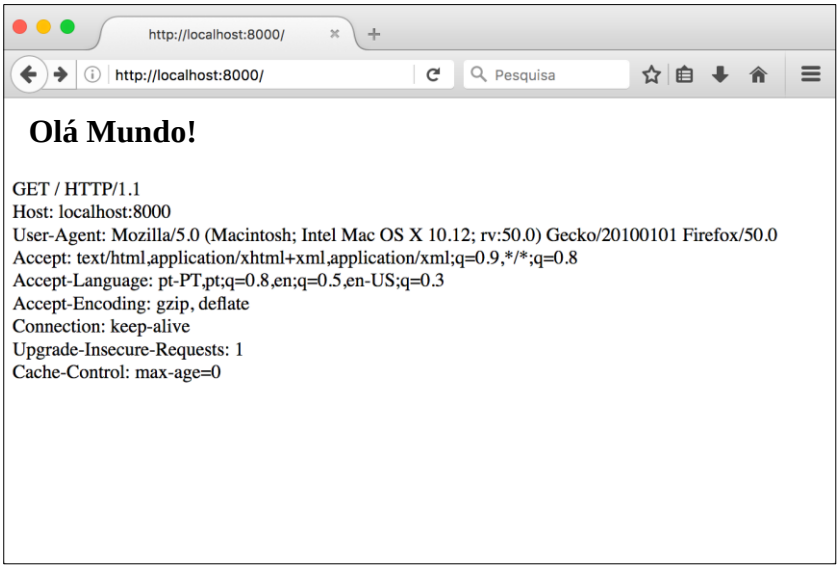
Leia o código do ficheiro *httpserver.py*, tente compreender como está estruturado e como funciona, e **verifique que na consola do IDE consegue ver os pedidos HTTP** recebidos cada vez que faz *refresh* no browser.

Nível 1

Considere o código que cria a resposta HTTP, e a imagem seguinte:

1. Altere a resposta de modo a que a mensagem *Olá Mundo* apareça, no *browse*r, como cabeçalho na página. (Sugestão: procure sobre a *tag h1*).

2. Altere a resposta de modo a acrescentar à mensagem o conteúdo do pedido feito pelo *browser* (na variável *request*). O conteúdo do pedido deverá ser apresentado bem formatado como na imagem. (Sugestão: considere fazer *split* do *request* pelas várias linhas "\n", e concatenar as linhas com a variável *response*, ou usar a tag *pre*).



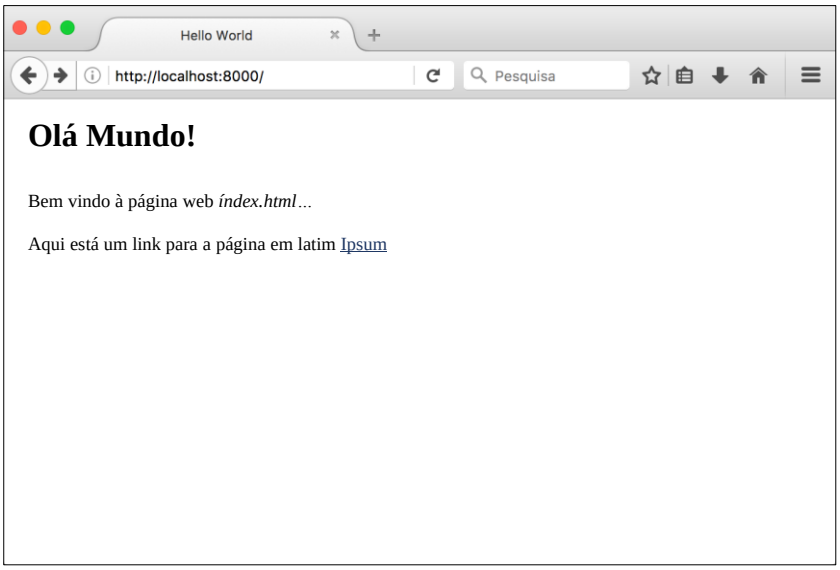
Programação Web
2019 / 2020
MEIT
Prática: Servidor de HTTP/1.0

Nível 2

Por defeito, quando um browser envia um pedido para a raiz do servidor (pedido tipo "*GET / HTTP/1.1*"), assume-se que a resposta deverá incluir o conteúdo do ficheiro *index.html*.

Altere a resposta do servidor de modo a enviar o cabeçalho *HTTP* seguido do conteúdo do ficheiro *htdocs/index.html*. (Sugestão: pesquise como pode abrir um ficheiro e ler o conteúdo do ficheiro como texto. Não se esqueça de fechar o ficheiro após ler o conteúdo..)

```
response = 'HTTP/1.0 200 OK\n\n'
# abrir ficheiro
# response += conteúdo do ficheiro
# fechar ficheiro
```



Reinicie o servidor, faça *reload* no browser e verifique que obtém a página de boas vindas ao ficheiro *index.html*. Experimente clicar no link para o ficheiro *ipsum.html* (Ipsum). Justifique por que razão o link não abre a respectiva página?

Nível 3

Pretende-se generalizar o caso anterior para permitir que o browser responda aos pedidos das várias páginas na pasta *htdocs*. Para tal considere a seguinte função para lidar com o pedido do *browser*:

```
def handle_request(request):
```

```
# Obter a primeira linha do request (Ex: GET /ipsum.html HTTP/1.1)
# Fazer o split do conteúdo (Ex: ['GET', '/ipsum.html', 'HTTP..'])
# Abrir o ficheiro
# Ler o conteúdo
# Fechar o ficheiro
# Retornar o conteúdo
```

1. Inicialmente crie a função *handle_request* para apenas imprimir o conteúdo do *request*. No ciclo principal do servidor, após receber o pedido do cliente, chame esta função invés de imprimir o *request*.

2. Modifique a função *handle_request* de modo a separar o *request* por linhas e imprimir na consola apenas a primeira linha (que contém o GET). (Sugestão: considere fazer *split* pelo carater '\n' e colocar o resultado do *split* numa variável de nome ***headers***).

3. Utilize o método *split* na primeira linha dos *headers* (a que contém o GET) de modo a separar o conteúdo da linha pelos vários componentes do GET. Ou seja, a partir de "*GET /ipsum.html HTTP/1.1"* deverá obter a seguinte lista: ['GET', '/ipsum.html', 'HTTP/1.1']. O nome do ficheiro a obter estará na segunda posição da lista.

4. Obtenha o nome do ficheiro a partir da lista anterior, e modifique a função de modo a retornar o conteúdo do ficheiro. Garanta que obtém o ficheiro a partir da pasta *htdocs*, e que se o nome do ficheiro for apenas "/", retorna o ficheiro *htdocs/index.html*.

5. Altere o código no ciclo principal do servidor de modo a retornar o conteúdo do ficheiro que obteve a partir da função *handle_request*.

```
while True:

    # Handle client request
    request = client_connection.recv(1024).decode()
    content = handle_request(request)

    # Envia a resposta HTTP
    response = 'HTTP/1.0 200 OK\n\n'
    response += content
```

Execute o servidor e teste se consegue abrir os ficheiro *index.html* e *ipsum.html*.

Nível 4

Experimente abrir uma página que não existe no servidor (ex: http://localhost:8000/teste.html). O servidor deverá dar erro e terminar, mas deveria retornar uma mensagem de erro.

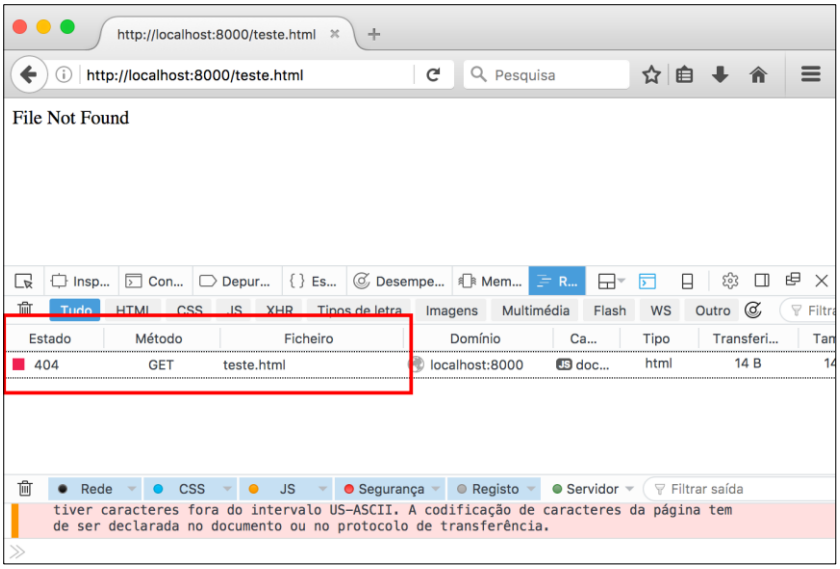
1. Modifique o código da função *handle_request* de modo a retornar a mensagem "Ficheiro não encontrado!" quando o ficheiro pedido via GET não existir na pasta *htdocs.*

```
def handle_request(request):
    ..
    try:
        # Abre o ficheiro
        # Lê o conteúdo
        # Fecha o ficheiro
        # Retorna o conteúdo
    except FileNotFoundError:
        return "Ficheiro não encontrado!"
```

Teste o funcionamento do servidor para ficheiros não existentes. Garanta que o serviço continua a funcionar para os ficheiros que existem.

2. Modifique o código no ciclo principal do servidor de modo a verificar se o ficheiro não foi encontrado e, nesse caso, enviar a resposta "HTTP/1.0 404 NOT FOUND\n\nFile not found". Se o ficheiro existir, deve continuar a enviar o estado 200 Ok.

Teste o funcionamento do servidor, e utilize as ferramentas de programador do seu *browser* de modo a verificar que está receber o *status* 404 correctamente.



(fim de enunciado)