Computação Paralela e Distribuída
2023 / 2024

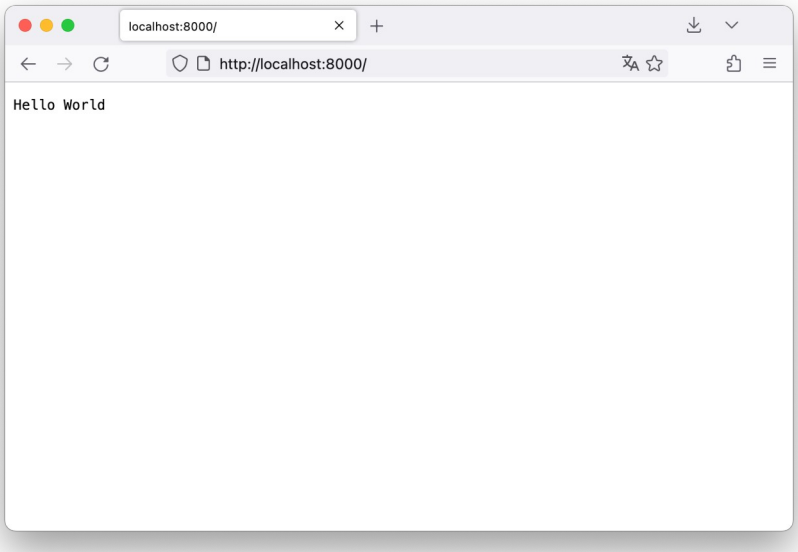
Licenciatura em Engenharia Informática

Lab. 08 – HTTP Server

Introdução

Nesta ficha iremos criar um pequeno servidor que implementa parcialmente o protocolo HTTP/1.0, usando *sockets* TCP/IP. Este laboratório tem por base o ficheiro **8-httpserver-base.zip** que contém uma implementação inicial do servidor e vários ficheiros *html* dentro da pasta *htdocs*.

Crie um projecto no seu IDE com estes ficheiros, e execute o código do servidor (*httpserver.py*). Execute o *browser* do seu computador e abra a página "http://localhost:8000/" (Linux ou MacOS) ou "http://127.0.0.1:8000/" (Windows). Se tudo estiver correcto, deverá ver o seguinte conteúdo no browser.



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.

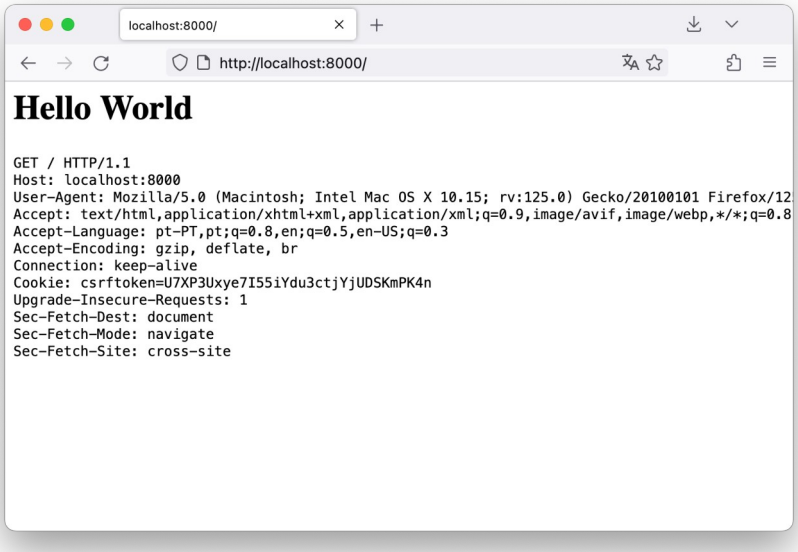
Nível 1 – Resposta HTTP

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

1. Altere a resposta de modo a que a mensagem *Hello World* apareça, no *browse*r, como cabeçalho na página. Utilize 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. Use a *tag <pre>*.

```
response = b'...<h1>Hello World</h1><pre>' + request.encode() + b'</pre>'
```

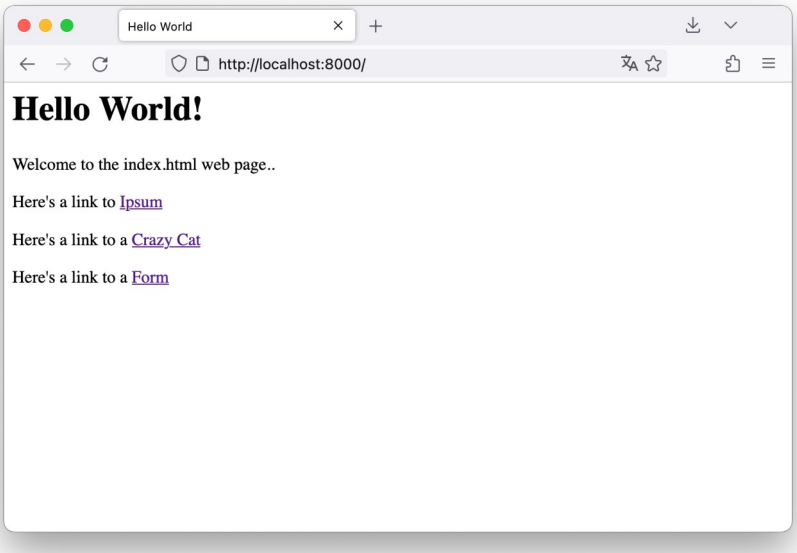


Nível 2 – Conteúdo de ficheiros

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*.

```
def handle_request(request):
    response = b'HTTP/1.0 200 OK\n\n'
    # with open(ficheiro, 'rb') as f
    #     response += f.read()
```



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). Por que razão o link não abre a respectiva página?

Nível 3 – Routing

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):
    # Get the lines of the request (Ex: [GET /ipsum.html HTTP/1.1], [...])
    # Split the content of first line (Ex: ['GET', '/ipsum.html', 'HTTP..'])
    # with open(file, 'rb')
    #     response += content
    # Return response
```

1. 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 caracter '\n' e colocar o resultado numa variável de nome ***lines***).

2. Utilize o método *split* na primeira linha das *lines* (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.

3. 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*. (Nota: se estiver a trabalhar em Windows, poderá ter de usar a barra invertida "\\" como separador de pastas).

Execute o servidor e teste se consegue abrir os vários links.

Nível 4 – Erros

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.

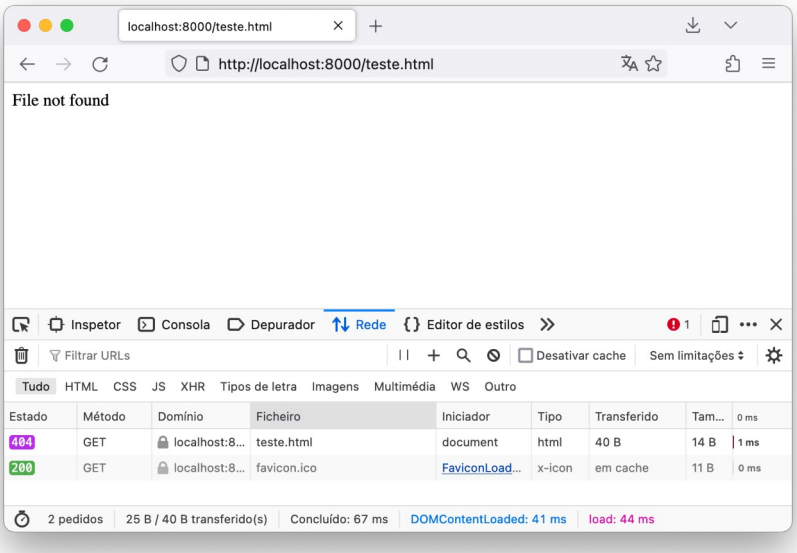
1. Modifique o código da função *handle_request* de modo a imprimir a mensagem “File not found!”, na consola, quando o ficheiro pedido via GET não existir na pasta *htdocs*.

```
def handle_request(request):
    ..
    try:
        # Open the file
    except FileNotFoundError:
        print(“File not found!”)
```

   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.



Nível 5 – Formulário

Abra o link do formulário e insira um nome e idade. Submita o formulário e note que a resposta será “File not found”. Isto acontece porque o formulário envia os dados via POST para um url de nome “handle_form”, mas não existe nenhum ficheiro com esse nome.

1. Na função *handle_request*, para além do nome do ficheiro (ou *url*) obtenha também o método HTTP (get, post, etc..) e imprima o *request* na consola

```
if method == 'POST':
    print(request)
    return b'HTTP/1.1 200 OK\n\n<pre>' + request.encode() + b'</pre>'
```

2. A informação enviada pelo *browser* deverá vir num formato chamado *www-form-urlencoded* e está no fim do pedido. Visto que já tem a lista com as linhas, obtenha a última linha com a informação. A informação será algo do género:

```
name=Manuel&age=23
```

3. Faça split pelo “&” e depois pelos símbolos “=” para obter o nome e idade da pessoa.

4. Por fim retorne, como resposta ao POST, um excerto de HTML com o nome e idade da pessoa.

```
content = f"name: {name} <br/> age: {age}".encode()
return b'HTTP/1.1 200 OK\n\n<h1>POST</h1>' + content
```

5. Teste com outros nomes e idades e verifique que continua tudo a funcionar correctamente.

(fim de enunciado)