Computação Distribuída
2019 / 2020

Licenciatura em Engenharia Informática

Trabalho Prático #3 – Servidor de HTTP/1.1

Introdução

Neste trabalho prático pretende-se construir um servidor HTTP utilizando os *sockets* TCP/IP. Este servidor irá implementar algumas das funcionalidades mais relevantes do protocolo HTTP/1.1.

Após iniciar, o servidor deverá aguardar pela ligação dos clientes. Estes clientes serão essencialmente *browsers*, mas poderão ser quaisquer outras aplicações que reconheçam o protocolo HTTP/1.1. Para mais detalhes sobre o protocolo HTTP, aconselha-se a leitura dos seguintes documentos:

- *HTTP Made Really Easy*: http://www.jmarshall.com/easy/http/
- *Especificação do protocolo HTTP/1.1*: https://www.w3.org/Protocols/rfc2616/rfc2616.html

É importante garantir que o servidor é capaz de lidar com múltiplas conexões em paralelo e que respeita a semântica de conexões HTTP/1.1, nomeadamente manter o *socket* aberto após o envio de cada resposta e terminar a ligação apenas quando o cliente o requerer.

Este trabalho tem por base o código fornecido no *Github* e é constituído pelos seguintes ficheiros:

- *httpserver.py* → Contém o servidor e *threads* nas classes *HTTPServer* e *HTTPConnection*.
- *tests.py* → Alguns unittests que validam o funcionamento correcto do servidor.
- *htdocs* → Pasta que contém alguns recursos como páginas HTML, imagens e outros.

Funcionalidades

Segue uma lista de requisitos que deverão ser implementados. Aconselha-se a atenção aos detalhes, como por exemplo, a utilização dos *status codes* apropriados, etc:

1. **Obtenção de recursos**: Deverá ser possível ao servidor retornar todas as páginas *html*, imagens, audio e vídeo localizados dentro da pasta *htdocs*.

2. **Status codes apropriados:** O servidor deverá suportar e retornar correctamente os códigos de status “200 Ok”, “400 Bad Request”, “403 Forbidden” e “404 Not Found”. Assuma que todos os ficheiros dentro da pasta “/private/” são privados e deverão retornar o 403.

3. **Cabeçalhos HTTP:** O servidor deverá reconhecer e utilizar sempre os cabeçalhos HTTP mais relevantes, nomeadamente o *Content-Length* e o *Content-Type*.

4. **Fecho de ligações:** O servidor deverá fechar ligações quando o cliente enviar o cabeçalho "*Connection: close*". Caso contrário, deverá manter a conexão com o cliente activa.

5. **Suporte ao método POST:** Para além do método GET, o servidor deverá também lidar com o método POST em duas situações:

   a) Quando o URI for do tipo ***/form***, o servidor deverá receber os dados em formato *application/x-www-form-urlencoded* e deverá, como resposta retornar os mesmos dados em formato *application/json*.

   Usando por exemplo o formulário em *htdocs/public/form.html* o servidor deverá retornar o json: *{"firstname": "Hello", "lastname": "World"}*. Note que os dados a transformar poderão ser genéricos, mas serão sempre do tipo name1=value1&name2=value2&...

   b) Quado o URI for do tipo /***json***, *o servidor deverá receber os dados em formato application/json* e retornar como resposta os mesmos dados no mesmo formato.

   Usando o exemplo em *htdocs/public/json.html*, o servidor deverá retornar o mesmo *json*. Note que o servidor deverá receber e retornar qualquer *json* que lhe seja enviado, e retornar *403 Bad Request* se o *json* for inválido.

6. **Segurança:** O servidor não deverá retornar ficheiros fora da pasta *htdocs* por uma questão de segurança (https://en.wikipedia.org/wiki/Directory_traversal_attack).

7. **Log:** Por cada pedido efectuado deverá guardar a data, hora, método HTTP, *url* e status code da resposta num ficheiro com o nome "log.txt". Alguns exemplos:

```
2020-05-12 09:33:36 GET /public/ipsum.html - 200 Ok
2020-05-12 09:33:36 GET /error.html - 404 Not Found
2020-05-12 09:33:36 GET /private/file.html - 403 Forbidden
2020-05-12 09:33:56 POST /form - 200 Ok
```

Unittests

O ficheiro *tests.py* contém um conjunto de *unittests* **incompleto** cujo objectivo é testar o funcionamento correcto do servidor em relação ao protocolo HTTP/1.1.

Alguns destes testes são, por exemplo, o teste *TestHTMLFiles.test_index_status* que verifica se o servidor retorna *200 Ok* quando se tenta obter o *index.html*, e o *TestHTMLFiles.test_index_content* que verifica se o servidor retorna correctamente algum do conteúdo do ficheiro *index.html*.

Notem que muitos dos testes estão incompletos (especialmente os que têm *pass*). O preenchimento correcto desses testes será cotado como um extra! Deverão mencionar no que testes implementaram.

Não modifiquem nada na classe *HTTPServer* que impeça a execução dos testes.

Entrega e avaliação

Este trabalho deverá ser realizado em **grupos de 2 alunos** usando o *Github Classrooms* como repositório de código e tem como data limite o **dia 07/Junho/2020 às 23h55**. Deverá ser modificado o *readme* de forma a incluir a identificação dos alunos e do docente responsável, o *link* para o repositório, e os extras implementados.

Todos os ficheiros deverão ser colocados num **ficheiro zip** (com o número dos elementos do grupo) e submetido via *moodle.*

Irá considerar-se a seguinte grelha de avaliação:

Correcção da solução (testes e avaliação manual)	12 val.
Qualidade e modularização do código (pylint, etc.)	04 val.
Extras (~1 valor por extra)	04 val.

Alguns possíveis extras a considerar: (1) preenchimento dos *unittests* incompletos – devem mencionar no *readme* que testes implementaram; (2) autenticação para aceder ao conteúdo da pasta "private", (3) mecanismo de *timeout* – i.e, fecho da ligação no servidor após alguns segundos de inactividade; (4) streaming de vídeo; (5) outros métodos HTTP relevantes, como *HEAD*; (6) utilização de mecanismos de *cache* – guardar em memória as últimas 2/3 páginas *html* ou ficheiros multimédia e devolver o seu conteúdo directamente da memória invés de ler do disco; (7) etc.., sejam criativos!

Nota final: Os alunos deverão implementar todas as funcionalidades mencionadas de forma a que todos os ficheiros e páginas *html* na pasta *htdocs* (incluindo o formulário e o *json*) funcionem como esperado num ambiente de um *browser.*

Bom trabalho!