Universidade Federal do Rio Grande do Norte
Departamento de Engenharia de Computação e Automação
DCA0130 – Redes de Computadores

Trabalho prático – Criação de um servidor HTTP

Instruções:

- O trabalho pode ser realizado duplas;
- A nota deste trabalho é parte da avaliação da Unidade II;
- O trabalho consistirá na implementação de um servidor HTTP simples;
- Não é permitido que sejam utilizadas bibliotecas "prontas" do protocolo HTTP. Tudo deve ser implementado "à mão".

Objetivo:

O objetivo deste trabalho é colocar em prática os conhecimentos apresentados em sala de aula a respeito da camada de aplicação e o protocolo HTTP. Vocês irão criar um servidor HTTP que deverá responder ao comando `GET` enviado por um cliente. O servidor deverá responder pedidos via telnet quanto via navegador. A implementação deverá ser em linguagem Python e vocês terão como base inicial o arquivo disponibilizado em:

Para Python 2.x:
`http://www.dca.ufrn.br/~viegas/disciplinas/DCA0130/files/Sockets/HTTP%20Server/HTTPserver.py`

Para Python 3.x:
`http://www.dca.ufrn.br/~viegas/disciplinas/DCA0130/files/Sockets/HTTP%20Server/HTTPserver.py3.py`

Forma de avaliação:

- O trabalho iniciará em sala de aula, mas poderá ser concluído em horário extraclasse;
- Cada dupla irá explicar ao professor em sala de aula (próximas aulas) como é o funcionamento da aplicação, bem como o código fonte/implementação;
- O código base DEVE ser mantido, seguindo a mesma lógica de implementação;
- CÓPIAS NÃO SÃO ADMITIDAS. Evitem usar implementações prontas ou de outros.

Ferramentas necessárias:

- Compilador python 2 ou 3, disponível em http://www.python.org/downloads
- Editor de texto (ex: Notepad++, gEdit) ou IDE de programação (ex: Eclipse, pyCharm)
- Terminal para a execução do python

Tarefas:

1. Execute o servidor HTTP fornecido e entenda o seu funcionamento:
   a. Abra um navegador e acesse o endereço do servidor:
      `http://127.0.0.1:8080`
      Repare que a página exibirá sempre a mensagem "Hello, World!"

   b. Abra um terminal e acesse por meio de `telnet`:

```
telnet 127.0.0.1 8080
GET / HTTP 1.1
```

Após o GET, dê dois "Enter". Será exibido o cabeçalho da mensagem (`HTTP/1.1 200 OK`) e também "Hello, World!". Repare que independentemente do comando digitado, o servidor retornará sempre a mesma resposta;

2. Após realizar os testes acima, analise o código fonte da aplicação com o uso de um editor de texto ou IDE de programação;

3. Faça as alterações necessárias no código para que o servidor seja capaz de processar um pedido GET de um cliente e retornar um arquivo .html para o mesmo;
   a. Crie um arquivo `index.html` (na mesma pasta em que o servidor estiver sendo executado) com o código abaixo para ser usado como resposta:

```
<html>
   <head><title>Este é o meu servidor!</title></head>
   <body>
         <h1>Olá mundo!</h1>
         O meu servidor funciona!
   </body>
</html>
```

   b. Quando um pedido é feito ao servidor, a variável `request` (linha 35) recebe os dados solicitados. Esta variável deve ser inspecionada para que se possa analisar o que está sendo pedido. É a partir dela que se devem tratar os casos abaixo solicitados. DICA: Esta variável `request` pode ser segmentada em pedaços de informação menores e consultados por índice (vetor).

   c. A sintaxe do pedido GET deve seguir a especificada pelo protocolo HTTP:

```
GET /caminho HTTP/versão
```

   Por exemplo:

```
GET /arquivo.html HTTP/1.1
```

   Por exemplo (caso o arquivo esteja em uma pasta):

```
GET /pasta/arquivo.html HTTP/1.1
```

   Outro exemplo:

```
GET / HTTP/1.1
```

   Neste caso quando não é especificado o arquivo no pedido, o servidor deve procurar pelo `index.html`.

   IMPORTANTE: Ao fazer um pedido com GET, nunca se deve inserir o caminho do arquivo no sistema de arquivos do sistema operacional, pois o HTTP não sabe interpretar esse caminho (exemplo a não usar: C:\Documentos\pasta\arquivo.html e nem usar /home/usuario/arquivo.html).

Deve-se inserir apenas o caminho do arquivo em relação (caminho relativo) à pasta em que o servidor HTTP está rodando.

d. Caso o caminho solicitado não exista, o servidor deve retornar o código 404 - página não encontrada. Este retorno deve seguir a especificação do protocolo HTTP, onde são enviados dois comandos: um para ser interpretado apenas pelo navegador (primeira linha) e outro para ser lido pelo cliente (segunda linha em diante), como o exemplo abaixo:

```
HTTP/1.1 404 Not Found\r\n\r\n
<html>
      <head></head>
      <body>
            <h1>404 Not Found</h1>
      </body>
</html>\r\n
```

e. Caso algum outro comando diferente de GET seja digitado, o servidor deve tratar a exceção e retornar "comando desconhecido" e continuar aguardando por novos comandos. Neste caso, o erro deve ser:

```
HTTP/1.1 400 Bad Request\r\n\r\n
<html>
      <head></head>
      <body>
            <h1>400 Bad Resquest</h1>
      </body>
</html>\r\n
```

f. Recorde-se que caso a página seja realmente encontrada, o servidor deve retornar (na primeira linha) o comando `HTTP/1.1 200 OK\r\n\r\n`. E só em seguida retornar o conteúdo da página html;

g. Sempre que um cliente enviar um comando GET para o servidor, este deve imprimir na tela (do terminal) todo o comando enviado pelo cliente;

h. É importante lembrar que, por omissão, quando se faz um pedido diretamente à raiz (isto é: `GET / HTTP/1.1`), sem especificar uma página, o servidor deve retornar a página padrão (geralmente o `index.html`).

Atenção: O servidor nunca deverá ser encerrado. Apenas os clientes terão as conexões abertas por mais tempo ou encerradas logo após um pedido.

4. Faça os testes do funcionamento do servidor usando o `telnet`, bem como o navegador. É importante notar que o navegador faz mais de um pedido por vez, pois solicite um

arquivo `favicon.ico` (arquivo de ícone para a aba). É necessário tratar esta situação na implementação do servidor. A nota do trabalho depende disso.

5. Ao terminar, chame o professor para apresentar/explicar o trabalho. Além disso, submeta o código via SIGAA, na tarefa correspondente.