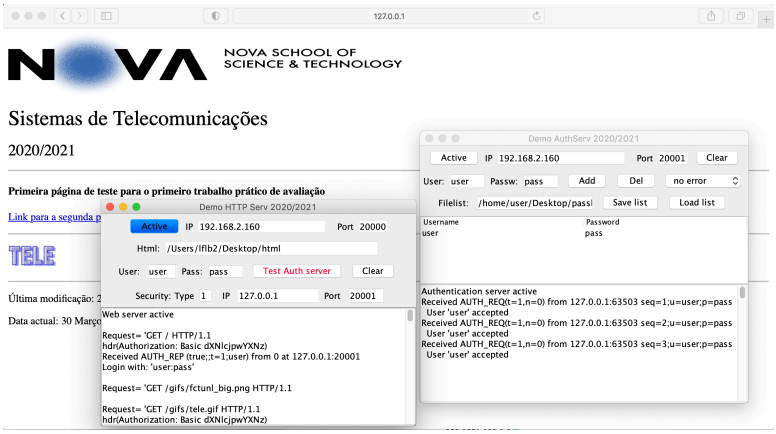
NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING



Sistemas de Telecomunicações

2020/2021

Trabalho 5:

Aplicação sobre sockets – serviço web com autenticação
PARTE 2 – Serviço web

Aula 6

Mestrado integrado em Engenharia Eletrotécnica e de Computadores

http://tele1.dee.fct.unl.pt **V1.0**

Luis Bernardo
Paulo da Fonseca Pinto

Índice

1. Objetivo 1
2. Especificações 1
2.1. Comunicação HTTP sobre um Socket Orientado à Ligação 1
2.2. Cache de palavras de passe validadas pelo serviço de Autenticação 3
3. Desenvolvimento do programa 4
3.1. Servidor web – Segunda semana 4
3.3. Avaliação do trabalho 6
Postura dos Alunos 6

1. OBJETIVO

Familiarização pelo aluno e verificação pelo docente da capacidade de usar *sockets* para comunicação entre máquinas e trabalhar com a interface de programação *sockets* do Java.

O trabalho consiste no desenvolvimento de

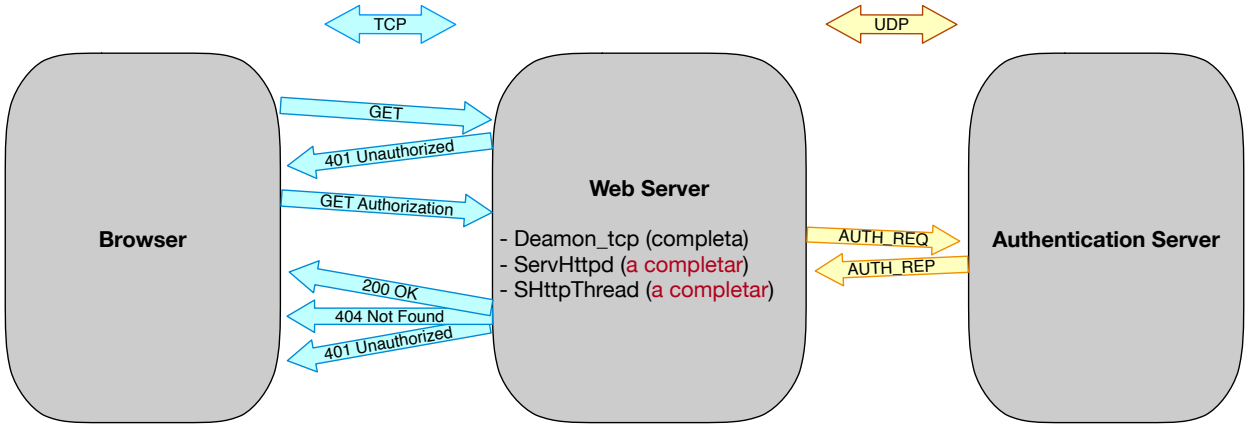
- um servidor de autenticação e de
- um servidor web

para criar um sistema onde o acesso às páginas web requer o conhecimento de uma palavra de passe e, portanto, só pode ser feito por um utilizador registado no serviço de autenticação. O servidor web comunica com o servidor de autenticação usando *sockets datagrama* de forma a validar as palavras de passe recebidas dos utilizadores. Para além disso, guarda numa lista local (*cache*) as palavras de passe durante um intervalo de tempo, podendo usá-las para validar utilizadores sem ter de voltar a perguntar ao servidor de autenticação. Os utilizadores usam browsers e comunicam com o servidor web pelo protocolo *HyperText Transfer Protocol* (HTTP).

O trabalho deve ser feito nas aulas de laboratório, sendo dividido em duas aulas: trabalho 4 e trabalho 5. Atendendo à situação pandémica os dois trabalhos serão avaliados presencialmente na aula do trabalho 5.

2. ESPECIFICAÇÕES

Em termos de especificações gerais, relembra-se apenas a figura em baixo do enunciado do Trabalho 4. Este trabalho foca na troca de mensagens a azul (à esquerda na figura).



2.1. COMUNICAÇÃO HTTP SOBRE UM SOCKET ORIENTADO À LIGAÇÃO

A comunicação entre um *browser* web e um servidor web faz-se através de um socket orientado à ligação, através da troca de mensagens de texto (eventualmente com conteúdos binários embebidos) que obedecem ao protocolo HTTP. Este protocolo suporta a transferência de ficheiros entre computadores. Neste trabalho vai-se usar apenas um pequeno subconjunto deste protocolo da camada aplicação, na sua versão mais simples (HTTP/1.0 e 1.1).

2.1.1. Endereçamento de páginas web

Uma página web é endereçada através de um localizador uniforme de recurso (*Uniform Resource Locator* = URL), composto por três partes:

http://tele1.dee.fct.unl.pt/st_2020_2021/pages/default.html

- http – identifica o protocolo usado;
- tele1.dee.fct.unl.pt – identifica o endereço IP do servidor. Este campo também pode ter o formato alternativo 172.16.54.1:20000, identificando o endereço IP e o número de porto explicitamente;
- /st_2020_2021/pages/default.html – identifica um ficheiro dentro do servidor.

2.1.2. Comunicação HTTP

Quando recebe um URL, um *browser* abre uma ligação TCP para o endereço e porto contidos no URL (ou para o porto 80, se este não for indicado), e envia uma primeira linha de texto terminada com a sequência de carateres "`\r\n`", com o seguinte conteúdo, a pedir o ficheiro referenciado no URL:

```
GET /st_2020_2021/pages/default.html HTTP/1.1\r\n
```

Para além desta primeira linha, o *browser* envia uma sequência de outras linhas de texto, terminadas com uma linha vazia (i.e., apenas com a mudança de linha). Cada linha não vazia contém um nome de uma propriedade seguida de dois pontos (':'), um espaço, e o valor da propriedade associada ao nome (e.g. *Host: tele1.dee.fct.unl.pt*). Neste trabalho apenas vai ser usada a propriedade "*Authorization*", com a informação de autenticação[1].

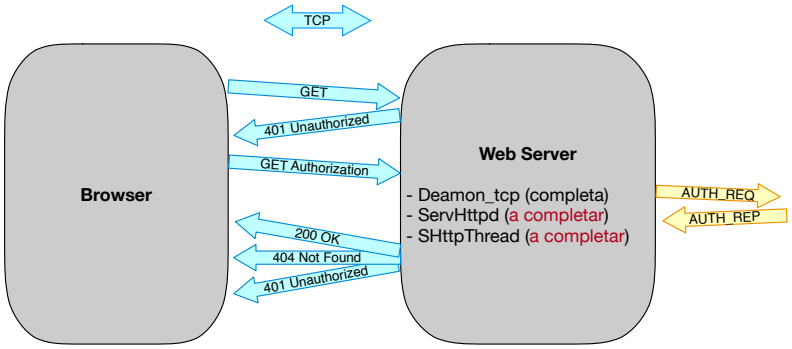
O servidor web pode saber qual é o ficheiro pedido lendo o conteúdo do segundo bloco da string recebida na primeira linha do pedido do browser. Caso contenha apenas "/", então deve ser devolvido o ficheiro correspondente a "`/index.htm`".

O protocolo HTTP especifica um formato de resposta, também constituído por uma sequência de linhas de texto legível, terminada por uma linha vazia (apenas com a mudança de linha "`\r\n`"), seguida do conteúdo do ficheiro. No entanto, a maior parte dos browsers modernos sabe interpretar o conteúdo do ficheiro pedido, mesmo que não seja enviada toda a sequência de linhas de texto especificada. Neste trabalho o servidor web deve enviar o cabeçalho de resposta com as linhas indicadas neste documento, em baixo. A ligação entre o browser e o servidor deve ser terminada após enviar o último octeto do ficheiro (quando é enviado), marcando dessa forma o fim do ficheiro para o *browser*.

2.1.2. Comunicação *browser*-servidor web

O browser envia pedidos ao servidor Web através do envio de pedidos HTTP, com a estrutura indicada na secção anterior. O suporte de autenticação de clientes numa ligação HTTP foi definido numa norma externa ao HTTP (IETF RFC 2617). Nessa norma são definidos dois modos de funcionamento: Basic e Digest. No modo Basic o nome de utilizador e palavra de passe são enviados codificados em base64, podendo ser facilmente descodificados por qualquer pessoa nas redes atravessadas pelos pacotes IP da ligação. No modo Digest, é utilizado um método criptográfico para cifrar estes dados. Neste trabalho vai ser realizado apenas o modo Basic, por ser mais simples.

[1] O protocolo HTTP é estudado em detalhe na disciplina Redes Integradas de Telecomunicações II, do 2º ciclo do MIEEC.



Como foi dito, um pedido transporta dados de autenticação para o servidor através da propriedade *Authorization*. Caso um pedido não contenha uma linha com esta propriedade, o servidor web deve requer a autenticação do cliente devolvendo o valor 401 (*Unauthorized*) na resposta e uma linha com *WWW-Authenticate* que indica o tipo de autenticação pretendido e o nome do domínio de segurança. No caso do trabalho, a mensagem de resposta para pedir a autenticação do browser deverá ser (onde `xxxxx` é o número de aluno):

```
HTTP/1.0 401 Unauthorized\r\n
WWW-Authenticate: Basic "ST 2020/2021 by xxxxx domain"\r\n
\r\n
```

Em consequência, o browser abre uma janela a pedir o nome de utilizador e a palavra de passe, e codifica em base64 a cadeia de carateres resultante da concatenação do nome (*user*) e da palavra de passe (*pass*), reenviando o pedido. Um exemplo de pedido, com o campo de autenticação com a *string* "*user:pass*" codificada seria:

```
GET /st_2020_2021/pages/default.html HTTP/1.1\r\n
Authorization: Basic dXNlcjpwYXNz\r\n
...
\r\n
```

Caso o pedido inclua uma autenticação válida, o servidor deve enviar a primeira linha de texto a confirmar que está a enviar o ficheiro que consiste na indicação do protocolo (`HTTP/1.0` – a versão mais simples), o código de sucesso (`200`) e numa informação textual associada ao código de sucesso ("`OK`"), seguida de uma outra linha com o nome do servidor e a linha vazia ("`\r\n`") a preceder o envio do ficheiro:

```
HTTP/1.0 200 OK\r\n
Server: ST 2020/2021 by xxxxx\r\n
\r\n
… { dados do ficheiro pedido } …
```

No caso de o ficheiro não existir, deve ser enviado um código de erro (e.g. 404 Not Found). Pode-se enviar uma página web a descrever o erro após este cabeçalho:

```
HTTP/1.0 404 Not Found\r\n
Server: ST 2020/2021 by xxxxx\r\n
\r\n
… {página web opcional}
```

2.2. CACHE DE PALAVRAS DE PASSE VALIDADAS PELO SERVIÇO DE AUTENTICAÇÃO

Quando o serviço de autenticação valida um par "*utilizador:palavra de passe*", este deve ser guardado em memória durante um intervalo de tempo que depende do número de protocolo

definido no trabalho 4 (de acordo com a tabela seguinte). Após o tempo expirar este registo é apagado e uma nova validação tem de ser feita junto do serviço de autenticação.

Caso não programe o protocolo com o parâmetro correto tem ZERO valores no exercício.

Validade								
`Val = 2s`	P=1	P=4	P=7	P=10	P=13	P=16	P=19	P=22
`Val = 4s`	P=2	P=5	P=8	P=11	P=14	P=17	P=20	P=23
`Val = 6s`	P=3	P=6	P=9	P=12	P=15	P=18	P=21	P=24

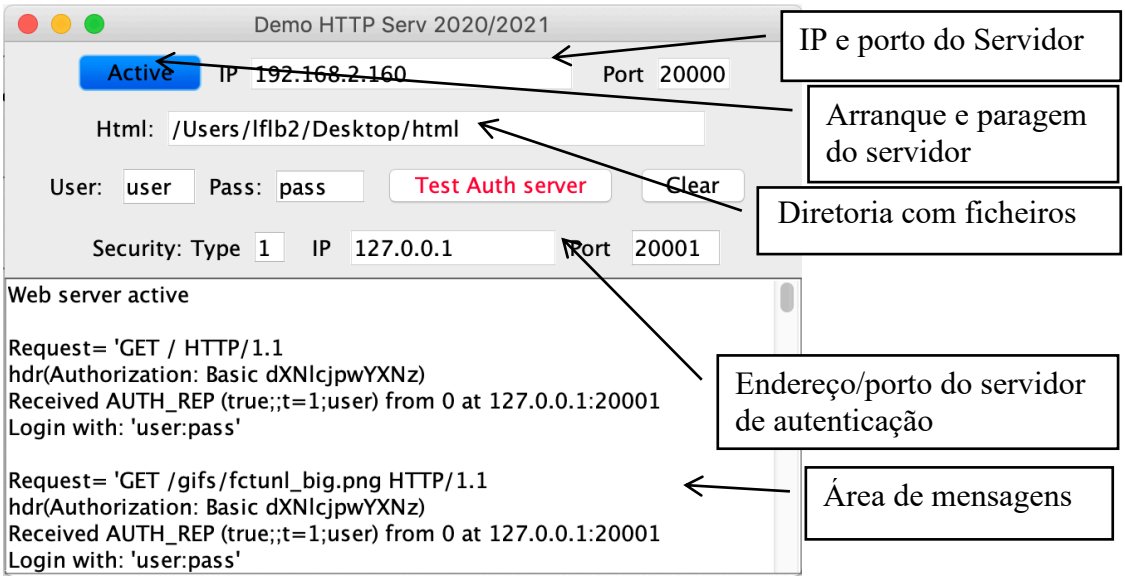
3. DESENVOLVIMENTO DO PROGRAMA

O programa está estruturado em duas aplicações. O servidor web é baseado no código de um servidor HTTP simplificado apresentado no documento tutorial, que já apresenta os mecanismos de leitura dos comandos mais importantes do protocolo HTTP. Este trabalho consiste em completar o código do servidor web, seguindo as instruções apresentadas neste documento.

Para desenvolver o código, deve continuar o projecto que desenvolveu no trabalho 4, que deve estar funcional.

3.1. SERVIDOR WEB – SEGUNDA PARTE

O servidor web (*ServHttpd*) responde a pedidos de um browser, enviando o conteúdo de ficheiros localizados numa diretoria especificada na interface gráfica. Valida o acesso a ficheiros utilizando o protocolo de comunicação com o servidor de autenticação, implementado no trabalho 4. Neste trabalho vai-se completar esta parte implementando a cache de palavras chave.



A interface gráfica permite especificar o porto TCP local (*Port*) (caso esteja ocupado, a aplicação vai incrementar esse valor até mais 100 para encontrar um livre. Se encontrar escreve-o na janela); a diretoria raiz (*Html:*) onde são lidos os ficheiros. Esta diretoria vai corresponder à raiz "/"; o endereço IP, porto e tipo usado com o servidor de autenticação (*Security:*).

Quando se prime o botão "*Active*" o servidor web cria um *ServerSocket* com o porto indicado em *Port* (ou superior), e lança a *thread Daemon_tcp* para ficar à espera de pedidos de browsers.

O programa fornecido é composto por três classes do pacote *server*:

- *Daemon_tcp.java* (completa) – *Thread* que recebe ligações no ServerSocket; cada nova ligação é tratada pelo objeto *SHttpThread*.
- *SHttpThread.java* (**a completar**) – *Thread* que processa os pedidos HTTP e envia a resposta; isto é feito dentro de uma ligação TCP.
- *ServHttpd.java* (**a completar**) – Classe principal com interface gráfica, que realiza a comunicação com o servidor de autenticação e a gestão da cache. Contém alguns métodos que podem ser úteis aos alunos, como *read_String*.

O programa fornecido realiza o processamento dos pedidos HTTP, faltando todas as tarefas relacionadas com a autenticação de acessos e com a gestão da cache de palavras de passe. As duas tarefas a realizar na classe *SHttpThread* são:

1. Programar parte da função *run* que recebe o pedido do browser e envia a resposta. O código fornecido recebe pedidos e responde com o código "200 OK" (enviando o ficheiro) ou "404 Not Found" se o ficheiro não existe. Pretende-se que acrescente o código para validar a palavra de passe, devolvendo o código "401 Unauthorized" se o utilizador não introduziu a palavra de passe, ou se esta está errada.

```
public void run();
```

Sugestão: Depois de ler a primeira linha de cabeçalho, use a função `parse_Authorization` para obter o valor do campo *Authorization* enviado pelo browser (*null* significa que não foi enviado). Use a função `return_unauthorized` para devolver o código 401 Unauthorized.

2. Programar a função *return_unauthorized* para responder ao browser com um código "401 Unauthorized".

```
public void return_unauthorized(PrintStream pout);
```

Sugestão: Consulte a secção 2.1.2 deste enunciado para saber qual é o formato da mensagem de resposta.

A tarefa a realizar na classe *ServHttpd* (a primeira parte já foi feita no trabalho 4) é:

3. Programar a função *validate_cached_passwd* para gerir a cache de palavras de passe. Deve declarar e inicializar uma variável da classe `HashMap<String,java.util.Date>` no construtor da classe. Cada vez que receber uma confirmação positiva do serviço de autenticação, deve usar a string que combina o nome de utilizado e palavara de passe (*user:pass*) como chave e a data atual como valor.
   Repare que não é tão fácil como parece, pois a palavra passe não vem no pacote `AUTH_REP`.
   Para simplificar, a verificação da validade de uma entrada na cache só é verificada da próxima vez que se quiser usar. Isto é, para cada invocação da função *validate_cached_passwd* deve procurar se a string já existe na cache, e se já existir, deve validar se já passou o tempo de vida, definido na secção 2.2 deste documento. No caso de já ter passado, a entrada deve ser retirada da cache. Neste caso, tem de se fazer a pergunta ao servidor de autenticação, e tem de ser colocada novamente na cache.
   Reparou na ajuda que lhe está a ser dada?

```
public boolean validate_cached_passwd(String url, String auth);
```

Sugestão: Converta as datas em inteiros (*long*) e calcule a subtração, para validar o tempo de vida das entradas na lista.

Os **pesos na nota final do trabalho das três tarefas** são respetivamente: **50%**, **10%** e **40%**.

3.3. AVALIAÇÃO DO TRABALHO

O trabalho é entregue no final da aula. Os estudantes devem:

1. No início da aula, pedir o número de protocolo (P) ao docente;
2. No final da aula, preparar um ficheiro comprimido com o código desenvolvido, com o nome, *00000-P2.zip* (ou *.tgz*), onde 00000 deve conter o número do estudante que fez o trabalho, e deve entregar esse ficheiro ao docente da aula prática;
3. Quando o docente lhe pedir, deve mostrar o código a funcionar, e estar preparado para responder a uma eventual pergunta que possa ser feita.

Nos casos onde existam dúvidas sobre a autoria do trabalho realizado (por exemplo, por vir todo realizado de casa antes da aula), pode ser marcada uma discussão posterior com o estudante.

Caso sejam detetados erros na realização, pode haver uma valorização parcial de cada elemento de avaliação, através da análise do código entregue.

POSTURA DOS ALUNOS

Cada aluno deve ter em consideração o seguinte:

- Não perca tempo com a estética de entrada e saída de dados
- Programe de acordo com os princípios gerais de uma boa codificação (utilização de indentação, apresentação de comentários, uso de variáveis com nomes conformes às suas funções...)