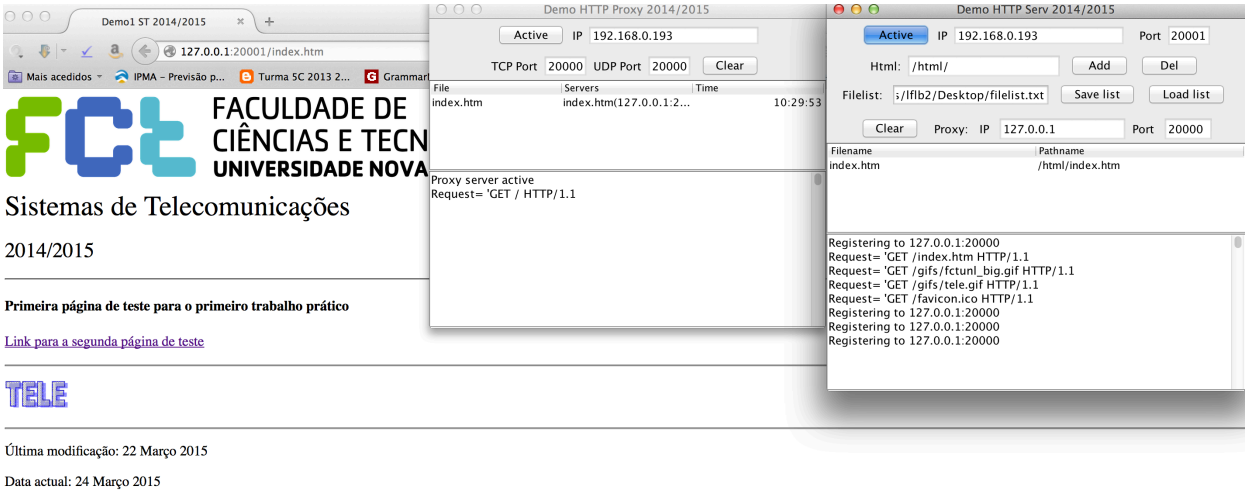
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Electrotécnica



Sistemas de Telecomunicações

2014/2015

Trabalho 1:
Aplicação sobre sockets – procurador web

Mestrado integrado em Engenharia Eletrotécnica e de Computadores

http://tele1.dee.fct.unl.pt

Índice

1. OBJETIVO

Familiarização com o uso de *sockets* para comunicação entre máquinas e com o funcionamento da interface de programação *sockets* do Java.

O trabalho consiste no desenvolvimento de um procurador (*proxy*) web e de um servidor web para criar um sistema em que o procurador distribui os clientes (*browsers web*) pelos vários servidores. Os servidores comunicam com o procurador usando sockets datagrama de forma a registar as páginas que contêm.

O trabalho deverá ser feito maioritariamente na aula prática, sendo dividido em duas partes, que serão avaliadas no final de cada uma das duas aulas práticas do trabalho.

> **Sugestões**: Em certas partes do enunciado aparece algum texto formatado de um modo diferente que começa com a palavra "Sugestões". Não é obrigatório seguir o que lá está escrito, mas pode ser importante para os alunos ou grupos onde ainda não haja um à-vontade muito grande com programação, estruturas de dados e algoritmia.

2. ESPECIFICAÇÕES

Pretende-se desenvolver um sistema de servidores web distribuídos gerido por um procurador. O procurador comporta-se como um portal para os clientes dado que eles têm de conhecer apenas o seu endereço e porto de acesso. Cada servidor web tem um porto e endereço IP não conhecido publicamente, que é fornecido pelos servidores ao procurador. A comunicação entre o cliente (*browser web*) e o procurador e o servidor usa o protocolo de aplicação *HyperText Transfer Protocol* (HTTP) sobre um socket orientado à ligação. A comunicação entre os servidores e o procurador é realizada através do envio periódico de pacotes datagrama com o registo dos ficheiros.

A aplicação servidor representa graficamente a listagem de ficheiros que publicita para o procurador, respondendo a pedidos de *browsers* pelos ficheiros locais. Também permite ao utilizador configurar qual é o procurador para onde envia os seus registos.

A aplicação procurador representa graficamente a lista de ficheiros que recebeu dos servidores, permitindo controlar os números de porto TCP e UDP usados.

Os clientes acedem ao serviço utilizando um *browser* web (Firefox, Google Chrome, Internet Explorer, Safari, etc.).

2.1. COMUNICAÇÃO HTTP SOBRE UM SOCKET ORIENTADO À LIGAÇÃO

A comunicação entre um *browser* web e um servidor 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).

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_2014_2015/pages/default.html

- http – identifica o protocolo usado;
- tele1.dee.fct.unl.pt – identifica o endereço IP. Este campo também pode ter o formato alternativo 172.16.54.1:20000, identificando o endereço IP e o número de porto explicitamente;
- /st_2014_2015/pages/default.html – identifica um ficheiro dentro do servidor.

Quando recebe um URL, um *browser* abre uma ligação TCP para o endereço e porto contido no URL (ou para o porto 80, se 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_2014_2015/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 (apenas com a mudança de linha), que não vão ser analisadas neste trabalho[1].

Um servidor ou procurador 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 uma 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 deve enviar as linhas indicadas neste documento, em baixo. A ligação entre o browser e o servidor deve ser cortada após enviar o último octeto do ficheiro, marcando dessa forma o fim do ficheiro para o *browser*.

2.1.2. Comunicação *browser*-servidor

Na comunicação entre o browser e o servidor, após receber o pedido representado acima, o servidor deve enviar a primeira linha de texto a confirmar que está a enviar o ficheiro a indicar o protocolo (`HTTP/1.0` – a versão mais simples), o código de sucesso (`200`) e uma informação textual associada ao código de sucesso ("`OK`"), seguida de uma linha com o nome do servidor e uma linha vazia ("`\r\n`") a preceder o envio do ficheiro:

```
HTTP/1.0 200 OK\r\n
Server: ST 2014/2015\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 2014/2015\r\n
\r\n
… {página web opcional}
```

2.1.3. Comunicação *browser*-procurador

O procurador não contém páginas web internamente. Tal como o servidor, também recebe o pedido dos browsers, mas caso saiba do ficheiro pedido, redireciona o browser para o servidor enviando a seguinte resposta ao *browser*:

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

```
HTTP/1.0 301 Moved Permanently\r\n
Server: ST 2014/2015\r\n
Location: http://IPserv:porto/st_2014_2015/pages/default.html\r\n
\r\n
```

Esta resposta devolve um código 301, a indicar que a página de moveu, e indica na terceira linha, qual é a nova localização onde o browser pode encontrar a página pedida. No caso do trabalho, a string `IPserv:porto` contém o endereço IP e o número de porto do servidor que vai tratar o pedido. Caso não seja conhecido um servidor com o ficheiro, o procurador deve enviar um código 404, tal como anteriormente.

A sequência de acessos que o *browser* faz até conseguir obter um ficheiro pedido será:

1. Criar ligação para procurador *web*, e pedido ficheiro;
2. Receber resposta com código 301 e URL do servidor com ficheiro (ou 404);
3. Criar nova ligação para servidor web com ficheiro, e pedido do ficheiro;
4. Receber resposta com código 200 e conteúdo do ficheiro, que é de seguida apresentado na interface gráfica.

2.2. COMUNICAÇÃO ENTRE SERVIDOR E PROCURADOR WEB

Quando se seleciona na interface gráfica a opção de partilhar um ficheiro, o servidor deve ativar um relógio e periodicamente deve enviar uma mensagem de registo (REGIST) para o procurador, com a seguinte estrutura:


```
Mensagem REGIST:// sequência de
 short type;     // Tipo de mensagem
 ...
seguido de uma sequência de campos que depende
do valor de type, e que inclui:
 short number1; // Número do aluno 1
 short number2; // Número do aluno 2, ou "0"
 short number3; // Número do aluno 3, ou "0"
 short len_name;// Comprimento do nome
 byte[] name;   // Nome do ficheiro
 int port;      // Numero de porto do servidor
```

type == 1	*type == 2*	*type == 3*	*type == 4*
short type=1; short ***number1***; short ***number2***; short ***number3***; short **len_name**; byte[] **name**; int **port**;	short type=2; short ***number1***; short ***number2***; short ***number3***; int **port**; short **len_name**; byte[] **name**;	short type=3; short **len_name**; byte[] **name**; int **port**; short ***number1***; short ***number2***; short ***number3***;	short type=4; int **port**; short **len_name**; byte[] **name**; short ***number1***; short ***number2***; short ***number3***;

O tipo de mensagem (*type*) a usar depende do número de protocolo que lhe vai ser fornecido pelo professor no início da aula da primeira aula de laboratório. O período de envio da mensagem REGIST também varia com o número de protocolo, conforme a tabela abaixo. **NO INÍCIO DA AULA VAI RECEBER O NÚMERO DE PROTOCOLO A IMPLEMENTAR. Caso não programe o protocolo com os parâmetros corretos tem ZERO valores no exercício.**

Período/Tipo	*type*= 1	*type*= 2	*type*= 3	*type*= 4
Period= 5s	P-1	P-2	P-3	P-4
Period= 8s	P-5	P-6	P-7	P-8
Period= 10s	P-9	P-10	P-11	P-12

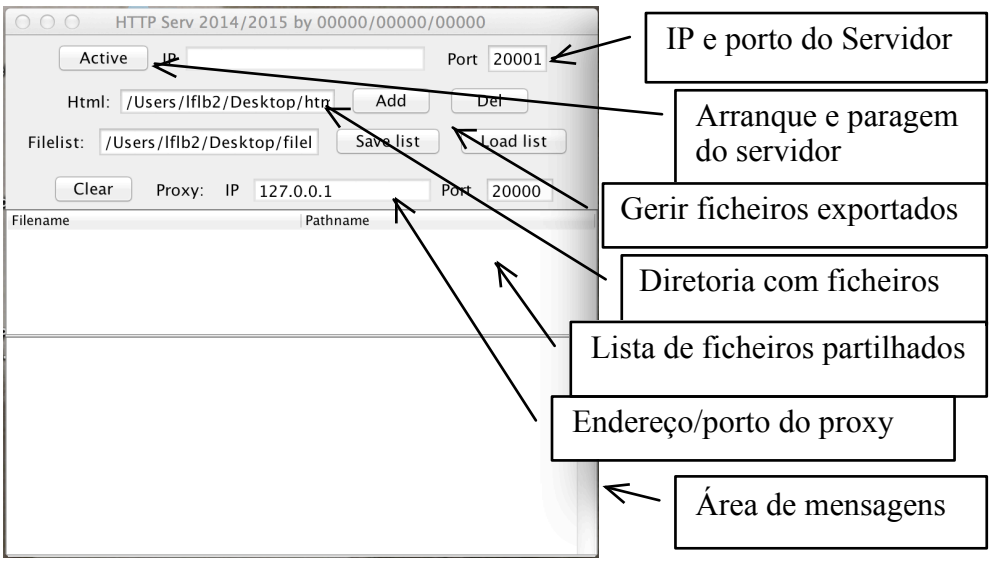
O procurador web recebe as mensagens de registo, guardando os registos dos ficheiro durante um período de tempo igual a 15 segundos. Caso não seja recebida nenhuma mensagem REGIST durante 15 segundos, o ficheiro deve ser excluído da lista.

3. DESENVOLVIMENTO DO PROGRAMA

O programa está estruturado em duas aplicações, baseadas no código de um servidor HTTP simplificado apresentado no documento "Introdução ao desenvolvimento de aplicações que funcionam em rede usando a linguagem Java", que já apresenta os mecanismos de leitura dos comandos mais importantes do protocolo HTTP. O trabalho consiste no completar do código fornecido das duas aplicações, seguindo as instruções apresentadas neste documento.

3.1. SERVIDOR WEB

O servidor web responde a pedidos de um browser, enviando o conteúdo de ficheiros. Mantém duas listas de ficheiros: todos os ficheiros acessíveis a partir de uma diretoria raiz especificada na interface gráfica; e uma lista de ficheiros introduzidos manualmente pelo utilizador, que vão ser exportados para o procurador através do envio de mensagens REGIST (uma mensagem por ficheiro).



A interface gráfica permite especificar o porto TCP local (*Port*) (caso esteja ocupado, a aplicação escolhe o primeiro livre acima do valor indicado); a diretoria raiz (*Html:*) onde são lidos os ficheiros; o endereço IP e porto do procurador (*Proxy:*); uma tabela com a lista de ficheiros partilhados com o procurador; botões para acrescentar (*Add*) e remover (*Del*) ficheiros à lista e para gravar (*Save list*) or ler (*Load list*) o conteúdo da lista de um ficheiro de índices (*Filelist:*). Quando se prime o botão "*Active*" o servidor web cria um *ServerSocket* com o porto indicado em *Port*, e ativa um temporizador de envio de registo, ficando preparado para responder a pedidos de browsers.

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

- *Daemon_tcp.java* (completa) – Thread que recebe ligações no ServerSocket;
- *SHttpThread.java* (completa) – Thread que processa os pedidos HTTP e envia a resposta (200 OK e o ficheiro, ou 404 Not found);
- *ServHttpd.java* (**a completar**) – Classe principal com interface gráfica, que faz a gestão do registo dos ficheiro no procurador.

O programa fornecido realiza todo o processamento dos pedidos HTTP, faltando todas as tarefas relacionadas com o registo de ficheiros no procurador. As três tarefas principais a realizar na classe *ServHttpd* (no primeiro dia do trabalho) são:

1. Programar a função *set_timer_function* e chamar a sua invocação para enviar todos os pacotes REGIST. Neste trabalho, a lista de ficheiros é gerida numa variável do tipo *Properties*, que funciona como uma *HashMap<String,String>*, com métodos para escrever em e ler de ficheiros. Para percorrer a lista de ficheiros, pode usar um iterador sobre a lista:

```
Iterator<String> it= files.stringPropertyNames().iterator();
```

2. Programar a função *send_Registration* para enviar uma mensagem REGIST compatível com o protocolo especificado no enunciado para o seu grupo:

```
public boolean send_Registration(String name);
```

3. Programar ao função *jToggleButton1ActionPerformed* (botão *Active*) para fechar o socket UDP e o timer, quando o botão é desligado.

Para além disso, também devem modificar a variável *server_name*, substituindo os 00000 pelos números dos elementos do grupo:

```
public final static String server_name = "HTTP Serv 2014/2015 by 00000/00000/00000";
```

A avaliação destas três tarefas é efetuada no final da primeira aula do trabalho, durante os últimos 25 minutos da aula, e os **pesos na nota final do trabalho das três tarefas** são respetivamente: **15%**, **30%** e **5%**.

3.2. PROCURADOR WEB

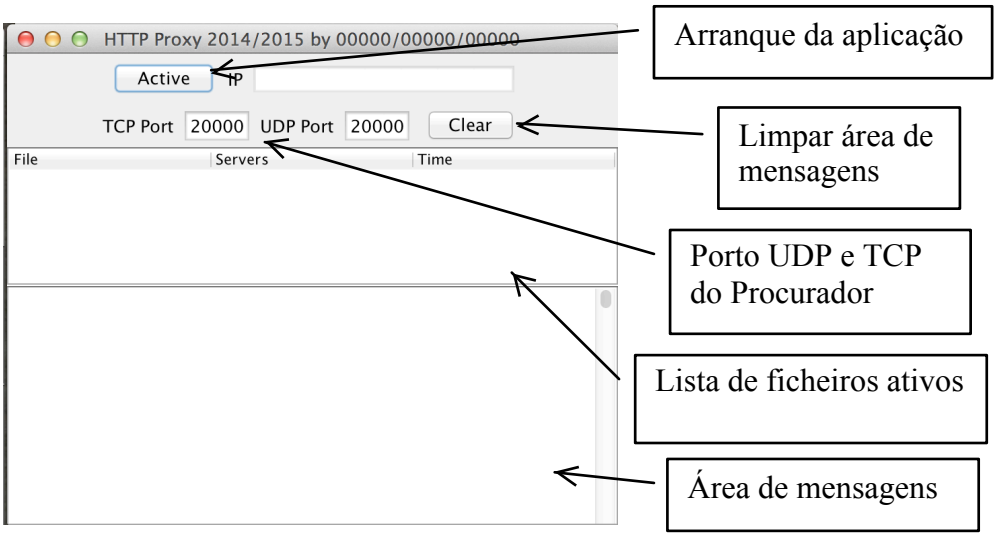
O procurador web recebe as mensagens REGIST num socket datagrama e guarda a informação recebida numa lista do tipo *HashMap<String,ServInfo>*, que é visualizada na interface gráfica. A classe *ServInfo* é fornecida com o enunciado e tem o seguinte conteúdo:

```
public class ServerInfo {
  public final int VALIDITY = 15000; // Validity = 15 seconds
  private final String name;         // Filename
  private final InetAddress ip;      // Server's IP address
  private final int port;            // Server's port number
  private final Date date;           // Registration date

  public ServerInfo(String name, InetAddress ip, int port); // Constructor
  public InetAddress ip();    // Return the server's IP address
  public int port();          // Return the server's port number
  public String name();       // Return the file's name
  public Date date();         // Return the registration date
  public boolean valid();     // Test if the registration is valid
  public String toString();
}
```

O procurador responde a pedidos de browsers enviando comandos de redirecção com o URL do último servidor que registou o nome pedido. Após 15 segundos (pode ser um pouco mais), deve remover os registos que deixaram de ser válidos.

A interface gráfica permite especificar os portos TCP e UDP locais (*TCP Port* e *UDP Port*). Quando se prime o botão "*Active*" o procurador web cria um *ServerSocket* com o porto indicado em *TCP Port*, um socket datagrama com o porto correspondente, ficando preparado para responder a pedidos de browsers e para receber registos de servidores.



O programa fornecido é composto por cinco classes do pacote *proxy*:

- *Daemon_tcp.java* (completa) – Thread que recebe ligações no ServerSocket;
- *Daemon_udp.java* (completa) – Thread que recebe datagramas no socket datagrama;
- *ServerInfo.java* (completa) – Estrutura de dados usada para guardar as mensagens do tipo REGIST numa lista;
- *PHttpThread.java* (**a completar**) – Thread que processa os pedidos HTTP e envia a resposta (301 Moved Permanently e o URL do servidor, ou 404 Not found);
- *ProxyHttpd.java* (**a completar**) – Classe principal com interface gráfica, que faz a gestão dos registos dos servidores.

O programa fornecido realiza todo o arranque dos sockets e quase todo o processamento dos pedidos HTTP, faltando realizar as tarefas relacionadas com o tratamento das mensagens REGIST e o envio do código de resposta 301 na comunicação com o browser web.

As duas tarefas a realizar na classe *ProxyHttpd* (no segundo dia do trabalho) são:

1. Programar ao função *receive_packet* para ler todos os campos da mensagem REGIST, guardando-a na lista *servers*, do tipo *HashMap<String, ServerInfo>*.

```
public synchronized void receive_packet(DatagramPacket dp, DataInputStream dis);
```

2. Programar ao função *remove_outdated_servers* para percorrer a lista *servers*, removendo todos os elementos que estão desatualizados. Acrescentar código para garantir que a função é chamada pelo menos de 2 em 2 segundos.

```
public void remove_outdated_servers();
```

Sugestões: Leia com atenção a definição da classe *ServerInfo*, nomeadamente do método *valid()*, antes de começar a realizar a tarefa 2.

As duas tarefas a realizar na classe *PHttpThread* são:

3. Programar ao função *return_redirect* para devolver a resposta HTTP do tipo 301 com o URL do servidor guardado no argumento *srv*.

```
private void return_redirect(PrintStream pout, String servername, ServerInfo srv)
```

4. Programar a função *run,* que processa os pedidos do browser, de forma a decidir se responde com um código 301 (redirecção) ou um 404 (não encontrado). Para tal, terá de descobrir se o ficheiro pedido está registado na lista *servers*.

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

Sugestões: Antes de realizar esta tarefa 4, leia com atenção o conjunto de métodos fornecidos pela classe *ProxyHttpd*. Para realizar esta tarefa não deve tornar a lista servers publica nem acrescentar novos métodos à classe *ProxyHttpd*.

Para além disso, também devem modificar a variável *server_name* na classe *ProxyHttpd* substituindo os 00000 pelos números dos elementos do grupo:

```
public final static String server_name = "HTTP Proxy 2014/2015 by 00000/00000/00000";
```

A avaliação destas quatro tarefas é efetuada no final da segunda aula do trabalho, durante os últimos 25 minutos da aula, e os **pesos na nota final do trabalho das quatro tarefas** são respetivamente: **15%**, **15%**, **10%** e **10%**.

Sugestão: Caso o grupo tenha mais do que um elemento, sugere-se que as tarefas 1-2 e 3-4 sejam realizadas em paralelo.

3.3. AVALIAÇÃO DO TRABALHO

A avaliação deste trabalho é realizada nos últimos 25 minutos das duas aulas onde se vai realizar o trabalho. Os alunos devem:

1. No início da primeira aula, pedir o número de protocolo ao docente;
2. No final de cada aula, preparar um ficheiro comprimido com o código desenvolvido para os dois projetos, com o nome, *00000-00000-00000.zip* (ou *.tgz*), onde 00000 deve conter os números dos alunos que fizeram o trabalho, e devem entregar esse ficheiro ao docente da aula prática;
3. Quando o docente vos pedir, devem mostrar o código a funcionar, e estarem preparados 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 os alunos do grupo.

Caso uma tarefa agendada para a 1º semana só seja realizada na semana seguinte, só é valorizado em 50%, a percentagem correspondente à tarefa na avaliação final. Caso seja implementado o protocolo errado, as tarefas realizadas são avaliadas com 0 valores.

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 grupo 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...) e
- Proceda de modo a que o trabalho a fazer fique equitativamente distribuído pelos dois membros do grupo.