ISEP – LEEC - Sistemas Computacionais 2017/2018

Ficha 8 – Comunicação via *sockets*

Esta ficha tem uma duração estimada de 2 semanas. A resolução dos exercícios II[1] e III-3) deverão ser entregues em manuscrito e individualmente dentro de 2 semanas, na aula PL respetiva.

Extraia o conteúdo do arquivo `ficha-sockets-ficheiros.zip` para um diretório à sua escolha.

I

1) Analise os programas `clt_exemplo.c` e `srv_exemplo.c`.
   1) Que dados deverão ser passados através da linha de comandos (vetor `argv[]`) para cada um dos programas? (procure as ocorrências de `argv`)
   2) Que dados são trocados entre os dois programas?
   3) Que serviço é fornecido pelo servidor `srv_exemplo`?
2) Abra uma *shell* e compile os programas.
3) Execute o programa `srv_exemplo`. O número do porto a indicar na linha de comando deverá ser superior a 1023 (ver nota de rodapé[2]).
4) Abra uma nova *shell* e execute o programa `clt_exemplo`. Como o servidor (`srv_exemplo`) está a ser executado na mesma máquina, o cliente pode indicar o endereço IP 127.0.0.1 (ou *localhost*) para aceder ao servidor.
5) Altere o servidor, recorrendo ao mecanismo de criação de processos (função `fork`()), de modo a que seja possível atender simultaneamente ligações de vários clientes. Para tal, deverá criar um novo processo por cada ligação aceite. Evite a acumulação de processos *zombie* através da chamada a `signal(SIGCHLD, SIG_IGN)`.

II

1) Implemente uma aplicação baseada no modelo cliente/servidor com ligação TCP/IP em que o cliente faz o envio do conteúdo de um ficheiro, indicado através da linha de comando, para o servidor. O servidor, por sua vez, deverá guardar os dados recebidos (i.e., o conteúdo do ficheiro enviado pelo cliente) no ficheiro criado pela seguinte instrução:

```
char filename[1024];
strcpy(filename, "fileXXXXXX.bin");
int fd = mkstemps(filename, 4);
```

   A função `mkstemps` cria e abre um ficheiro com nome único, pronto para operações de leitura e escrita. Se preferir trabalhar com as funções de entrada/saída da biblioteca standard de C, pode aplicar a função `fdopen` ao descritor de ficheiro `fd`.
   Para implementar a leitura e escrita dos ficheiros, pode basear-se no programa fornecido no ficheiro `simple_clone.c`.

[1] Pode omitir do manuscrito todo o código aproveitado dos ficheiros fornecidos, desde que devidamente mencionado.
[2] Só o utilizador *root* tem permissão para fazer o *bind* aos primeiros 1024 portos. Estes portos são reservados para serviços tipicamente encontrados nos servidores e estações de trabalho (servidor *Web,* serviço de sistema de ficheiros através da rede, etc.)

2) De forma análoga ao exercício I-5), altere o servidor, recorrendo ao mecanismo de criação de processos, de modo a que seja possível receber simultaneamente ficheiros de vários clientes.

III

O *Hypertext Transfer Protocol* (HTTP) é o protocolo de aplicação responsável pelas trocas de dados na *World Wide Web* (WWW), tais como o acesso a recursos *web* (páginas HTML, *scripts* PHP, imagens, etc.).
No caso do acesso a um recurso *web*, o utilizador começa por introduzir na barra de endereços do *web browser* (e.g., Chrome) o *universal resource locator* (URL) correspondente (e.g., `http://www.dee.isep.ipp.pt/~jes/sistc/pl/ola.html`). O URL, tal como a designação indica, identifica o recurso no universo da *Internet*, nomeadamente através do protocolo de acesso (http, no exemplo acima), nome do servidor (`www.dee.isep.ipp.pt`, no exemplo acima) e recurso propriamente dito no espaço de nomes do servidor (`~jes/sistc/pl/ola.html`, no exemplo acima)[3].
O HTTP assenta tipicamente no TCP/IP. O porto padrão para os servidores HTTP (também designados servidores *web*) é o 80, sendo assumido implicitamente no URL. Após estabelecer a ligação TCP/IP com o servidor HTTP, o *web browser* faz o pedido da página HTML usando o método GET do HTTP:

```
GET ~jes/sistc/pl/ola.html HTTP/1.1
Host: www.dee.isep.ipp.pt
```

(incluir linha em branco, ou seja, "`\r\n`")

A resposta do servidor deverá ter um conteúdo semelhante ao apresentado abaixo:

```
HTTP/1.1 200 OK
Date: Mon, 06 Feb 2017 09:27:29 GMT
Server: Apache/2.2.21 (Fedora)
Last-Modified: Wed, 03 Feb 2016 16:55:35 GMT
ETag: "a40df1-2e-52ae07c990fc0"
Accept-Ranges: bytes
Content-Length: 46
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Language: pt

<html>
<body>
Olá mundo!<br>
</body>
</html>
```

1) O programa `http_get.c` ilustra a transação descrita acima. Analise o código do programa (este código será necessário numa futura aula PL).
   a) Que tipo de comunicação (SOCK_STREAM/SOCK_DGRAM) é implementado neste programa?
   b) Em que tipo de aplicação este código seria normalmente usado, num *web browser* ou num servidor *web*?

2) Compile e execute o programa. Adicionalmente, aceda, num *web browser*, ao endereço `http://www.dee.isep.ipp.pt/~jes/sistc/pl/ola.html`. Compare os

[3] Para mais detalhes, consultar https://tools.ietf.org/html/rfc7230#section-2.7

resultados com o *output* do programa `http_get`. Ambos os programas (`http_get` e *browser*) recebem a mesma resposta do servidor. No entanto, o *browser* interpreta o código HTML para fazer a apresentação, devidamente formatada, do conteúdo da página.

3) Altere o programa de forma a omitir o cabeçalho do HTTP e fazer apenas a impressão do código HTML. Para tal, só deverá iniciar a impressão dos carateres após a receção de uma linha em branco (`"\r\n"`).
   Uma forma de abordar o problema será fazer a leitura linha a linha da resposta HTTP até que seja detetada a linha "em branco" (`strcmp(buffer, "\r\n")`). A leitura linha a linha pode ser implementada com a função `fgets`, após associar um *stream* ao descritor de *socket* através da função `fdopen`. Por exemplo:

```
FILE *fp = fdopen(s, "r+");
fgets(buffer, sizof(buffer), fp);
```

IV

Hoje em dia existem diversas implementações de servidores HTTP para as mais diversas finalidades, pelo que a implementação de raiz de um servidor HTTP raramente é necessária. O programa fornecido no ficheiro `dummy_webserver.c` faz-se passar por um servidor HTTP, mas implementa apenas um serviço rudimentar que consiste em devolver ao cliente uma página HTML com a reprodução da mensagem HTTP enviada inicialmente por este último.

a) Numa *shell*, compile e execute o programa:

```
make dummy_webserver
./dummy_webserver 8080
```

b) Aceda, num *web browser*, ao endereço `http://localhost:8080/ola.html` e observe a informação apresentada, nomeadamente a linha com o pedido GET. Relacione o resultado com o código do ficheiro `dummy_webserver.c`.

V

1) Implemente um programa que permita a receção de mensagens de texto através porto UDP 3000 e faça a sua impressão, assim como do endereço IP e porto do emissor, no ecrã.
2) Implemente um programa que permita enviar mensagens para o programa descrito no ponto 1 até que seja terminado com a combinação **ctrl+c**.
3) Verifique que o programa inicial pode receber sequências de mensagens de diferentes clientes sem recorrer a qualquer tipo de mecanismo multi-tarefa (criação de novos processos ou *threads*).
4) O tamanho máximo dos pacotes IPv4 é 65536 bytes. No entanto, em geral, a camada de ligação (e.g., a rede *ethernet*) não permite o envio de tramas dessa dimensão. O protocolo IPv4 suporta um mecanismo de fragmentação de forma a superar essa limitação. Contudo, esse mecanismo tem algumas desvantagens pelo que, em Linux, está desativado por omissão para o UDP. Verifique, por tentativa e erro (também pode basear-se na informação fornecida pelo comando `ifconfig`), qual o tamanho máximo que as mensagens podem ter para serem enviadas com sucesso.