ISEP – LEEC - Sistemas Computacionais 2016/2017

Ficha 4 – Comunicação via *sockets* e serviços *web*

Esta ficha tem uma duração estimada de 4 semanas. A resolução dos exercícios II e IV[1] deverá ser entregue em manuscrito e individualmente.

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

I

1) Analise os programas `clt_exemplo.c` e `srv_exemplo.c`.
   a) Que dados deverão ser passados através da linha de comandos (vetor argv[]) para cada um dos programas? (procure as ocorrências de argv)
   b) Que dados são trocados entre os dois programas?
   c) 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 ("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* (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.
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.

[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.)

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 protocolo HTTP assenta tipicamente nos protocolos 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)

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.
   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 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 altera a apresentação dos resultados de acordo com o código HTML recebido.
3) 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

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

uma página HTML com a reprodução da mensagem HTTP enviada inicialmente por este último. Numa *shell*, compile e execute o programa:

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

De seguida, aceda, num *web browser*, ao endereço http://localhost:8080/ola.html e observe a informação apresentada. Relacione o resultado com o código do ficheiro dummy_webserver.c.

IV

Pretende-se criar um programa em C para ambiente Linux que permita replicar, com base numa interface em modo de texto, os serviços oferecidos pela aplicação *web* desenvolvida na aula anterior. O esqueleto do programa é fornecido no ficheiro veic_clt.c, devendo ser completado com as chamadas ao servidor. Tal como na ficha anterior, cada registo é constituído por três campos: número de matrícula (6 caracteres), nome do proprietário (máximo de 80 caracteres) e valor comercial (valor numérico com um máximo de duas casas decimais). O programa fornecido inclui a definição do tipo de dados estruturado `vehicle_t` que deverá ser usado na declaração das variáveis necessárias para manipular os registos.

De forma a facilitar a implementação do novo cliente, pretende-se também que o servidor disponibilize as suas funcionalidades na forma de serviços *web* do tipo *representational state transfer* (REST). Esta interface de programação será implementada com base no código implementado para o *website*, nomeadamente os ficheiros vehicle.php e vehicles.php. No entanto, no caso do serviço de envio da lista de registos, pretende-se que a lista seja devolvida ao cliente no formato JSON[4], em vez do formato HTML usado para o website.

1) Verifique que o website que criou na ficha anterior se encontra operacional, através da consulta do seguinte URL:
   http://localhost/~sistc/subdiretorio_do_aluno/

2) Complete o código da opção de inserção no ficheiro veic_clt.c. Tal como na implementação do *website*, o serviço de inserção através do método POST. A inserção de registos deverá ser feita, tal como no formulário do *website*, através do URL http://servidor/~sistc/subdiretorio_do_aluno/vehicle.php. De forma a usar o método POST, a mensagem HTTP a enviar ao servidor deverá ter a forma apresentada no exemplo seguinte:

   ```
   POST /~sistc/subdiretorio_do_aluno/vehicle.php HTTP/1.1
   Host: localhost
   Content-Type: application/x-www-form-urlencoded
   Content-Length: 44
   <linha em branco>
   plate=11AA22&owner=John Jones&value=10000.00
   ```

   No exemplo acima, o valor 44, indicado para o campo `Content-Length` do cabeçalho HTTP, é o comprimento da *string* contendo a lista de parâmetros deste exemplo (última linha). Tal no caso do exemplo do método GET, tenha em conta que cada linha das mensagens do protocolo HTTP deverá ser terminada com a sequência "\r\n".

[4] http://www.json.org/

3) Em caso de sucesso na inserção do registo, o servidor retorna uma mensagem HTTP com o código 200:

```
HTTP/1.1 200 OK
```

Em caso de erro, dependendo do tipo de erro, a resposta do servidor apresentará um código numérico diferente. Adicionalmente, se o erro for detetado no código PHP, a resposta irá conter uma listagem em HTML com a descrição do erro (ver ficheiros vehicle.php e db_functions.php). A mensagem com a descrição do erro está embebida entre as anotações `<ws-error></ws-error>`[5].

Altere o código da opção de inserção do cliente de modo a notificar o utilizador em caso de erro. Caso a resposta do servidor inclua a anotação `<ws-error>`, o programa deverá imprimir a mensagem correspondente. Essa mensagem pode ser obtida recorrendo à seguinte sequência de instruções:

```
//assume que a resposta do servidor está armazenada na variável char buffer[]
char *ptr,*ptr2;
//procura a tag <ws-error>
if( (ptr = strstr(buffer, "<ws-error>"))!=NULL)
{
    //ptr fica a apontar para o primeiro carácter da mensagem de erro
    ptr = strstr(ptr,">") + 1;

    //função strsep substitui o '<' de "</ws-error>" pelo fim de string
    ptr2 = ptr;
    ptr = strsep(&ptr2, "<");

    printf("Error message from server: \"%s\"\n\n", ptr);
}
```

4) Altere o ficheiro vehicles.php, localizado em ~sistc/public_html/subdiretório_do_aluno/, de forma a enviar os dados em formato JSON (sem qualquer anotação HTML) no caso do parâmetro "o" tomar o valor "JSON". A codificação para o formato JSON pode ser feita através da função PHP json_encode[6].

5) Complete o código da opção de listagem no ficheiro veic_clt.c. Tal como na implementação do *website*, o serviço de listagem é acedido através do método GET. Do ponto de vista do cliente, a implementação do método GET deverá ser feita de forma análoga ao exemplificado no programa http_get desta ficha. O URL terá a seguinte forma:
http://servidor/~sistc/subdiretorio_do_aluno/vehicles.php?o=JSON.

Tal como indicado acima, o servidor envia os dados no formato JSON. A biblioteca padrão de C não inclui nenhuma função para manipulação de dados no formato JSON. Embora existam algumas bibliotecas de domínio público, tendo em conta a simplicidade dos dados, deverá implementar uma função especificamente para esta aplicação que permita recuperar os valores de cada um dos campos de cada registo e

[5] Estas anotações não fazem parte do HTML padrão, foram definidas especificamente para esta aplicação. A definição de anotações "proprietárias" é uma funcionalidade introduzida no HTML5.

[6] http://php.net/manual/en/function.json-encode.php

imprimi-los no ecrã. Abaixo, apresenta-se um exemplo de uma listagem em JSON contendo a informação de dois registos:

```
[["IJD007","John Doe","100000.00"],["00JJ77","Jane Doe","50000.00"]]
```

Uma possível implementação para a descodificação e apresentação da resposta do servidor é apresentada na Figura 1. Esse código deverá ser executado para cada carácter recebido.

Descrição do código da Figura 1:

A rotina apresentada deverá ser usada para analisar sequencialmente cada um dos caracteres da representação JSON do vetor de registos. Esta rotina lê os registos uma a um. Durante a leitura, os dados do registo são armazenados na variável v (do tipo vehic_t). Cada vez que a rotina verifica que terminou a leitura de um registo, chama a função vehic_print para fazer a impressão desse registo.

De forma a identificar os valores de cada um dos campos do registo, a rotina procura o carácter usado pelo JSON para delimitar *strings*, nomeadamente as aspas. A rotina usa a variável state para gerir essa pesquisa e para identificar o campo que está a ler em cada momento. A variável state pode tomar valores inteiros de 0 a 5, indicando os seguintes estados:

- Procura das aspas (início do campo "plate"), no estado 0.
- Leitura do campo *plate*, no estado 1.
- Procura das aspas (início do campo *owner)*, no estado 2.
- Leitura do campo *owner*, no estado 3.
- Procura das aspas (início do campo *value)*, no estado 4.
- Leitura do campo *value*, no estado 5.

Como pode ser visto, o código para os estados 0, 2 e 4 é o mesmo, pois em ambos os estados o objetivo é encontrar as aspas.
Nos estados 1, 3 e 5, a variável nchars é usada para determinar em que posição da variável deverá ser guardado o carácter recebido, assim como para garantir que o número de caracteres armazenados não excede o máximo previsto.
Em todos os estados, a deteção das aspas (abertura de aspas nos estados pares e fecho de aspas nos estados ímpares) provoca a passagem para o estado seguinte. No caso do estado 5, o estado seguinte é o 0.

6) Teste as funcionalidades do cliente criado.

```
char c; //should contain received character
vehic_t v;
char buffer2[256];
int nchars, state = 0;

    switch(state) {
    case 0:
    case 2:
    case 4:
      if( c == '"' ) { //found opening "
        state++;
        nchars = 0;
        break;
      }

      if( c == '['  && state != 0 ) { //new record before completing current one
        printf("invalid record\n");
        state = 0;
        break;
      }

      break;

    case 1://reading plate
      if( c == '"' ) {  //found closing "
        state = 2;
        break;
      }

      if( nchars == 6 ) //discard unexpected chars
        break;

      v.plate[nchars++] = c;
      break;

    case 3://reading owner
      if( c == '"' ) { //found closing "
        state = 4;
        v.owner[nchars] = 0;
        break;
      }

      if( nchars == VEHIC_MAXPLEN-1 ) //discard unexpected chars
        break;

      v.owner[nchars++] = c;
      break;

    case 5://reading value
      if( c == '"' ) { //found closing "
        state = 0;

        buffer2[nchars] = 0;
        v.value = atof(buffer2);
        vehic_print(&v);
        break;
      }

      if( nchars == sizeof(buffer2)-1 ) //discard unexpected chars
        break;

      buffer2[nchars++] = c;
      break;
    }
```

Figura 1 – Leitura de vetor de registos em formato JSON