Ficha de exercícios

(Perguntas de avaliação de anos letivos anteriores)

Tópicos abordados:
- *Bits* e *bytes*
- *Sockets* UDP e TCP
- Servidores UDP e TCP

Pergunta 1

(Escreva as suas respostas a esta pergunta no diretório **`"~/Ficha/R_NUMERO/Pergunta1"`**. Deve indicar o seu nome completo e número de estudante IPLeiria no ficheiro **README.txt** a ser criado no diretório)

NOTA 1: Não é permitida a chamada a comandos externos através da função *system* ou de outra com funcionalidade similar.

NOTA 2: A solução deve ser implementada com recurso a última versão do *template* **EmptyProject-Client-Server-Template**.

NOTA 3: Assuma que não existe perda de datagramas entre os programas cliente e servidor.

NOTA 4: Código entregue que **não compile** leva à atribuição da classificação de **0 (zero) valores** à resposta.

Uma instalação de domótica possui 8 dispositivos, que permitem controlar diversos parâmetros de uma habitação. O estado, de cada parâmetro, da habitação está codificado num inteiro de **16 bits** sendo a identificação dos dispositivos a seguinte:

N.º Dispositivo	Dispositivo/atuador
1	Portão da garagem (aberto / fechado)
2	Iluminação do *hall* de entrada
3	Iluminação sala
4	Iluminação jardim
5	Persiana 1
6	Persiana 2
7	Persiana 3
8	Piso radiante (ligado / desligado)

Se o *bit* correspondente a cada dispositivo estiver ativo (valor a 1), significa que o respetivo dispositivo se encontra ligado (ou aberto). Caso contrário, o dispositivo encontra-se desligado (ou fechado).

Recorrendo à linguagem C, elabore uma aplicação cliente/servidor, assente no protocolo UDP, que permita efetuar a gestão dos dispositivos da habitação. A aplicação (*domus*) deverá utilizar um protocolo aplicacional básico de pedido/resposta, onde cada pedido poderá ter um ou dois *bytes*, consoante o tipo. Assim, o primeiro *byte* do pedido é 0, 1 ou 2, indicado o qual o tipo. Para pedidos do tipo 1 ou 2 acrescenta-se um segundo *byte* que indica o número de dispositivo. Concretamente, o servidor deve suportar os pedidos indicados na tabela seguinte, produzindo a resposta explicitada na coluna "Resposta".

Código	Pedido	Resposta
`0`	Enviado para pedir o estado da instalação de domótica	Inteiro de **16 bits** representativo do estado da instalação de domótica
`1X`	Pedido para ligar/abrir o dispositivo. Envia-se o código 1, seguido do número do dispositivo (também um inteiro de 1 byte). O servidor deverá colocar a **um** o bit que que corresponde ao dispositivo X.	**0** – Indica que o dispositivo já se encontrava ligado/aberto **1** – Indica que o dispositivo foi ligado/aberto com sucesso **2** - Indica que o dispositivo não é válido
`2X`	Pedido para desligar/fechar o dispositivo. Envia-se o código 2, seguido do número do dispositivo X (também um inteiro de 1 byte). O servidor deverá colocar a **zero** o bit que corresponde ao dispositivo X.	**0** – Indica que o dispositivo já se encontrava desligado/fechado **1** – Indica que o dispositivo foi desligado/fechado com sucesso **2** - Indica que o dispositivo não é válido

A aplicação servidor (**domus_svc**) recebe como argumentos de entrada:

- Porto (--port) onde o servidor ficará à espera de pedidos;
- Inteiro de 16 bits com o estado inicial (--status) dos dispositivos.

A aplicação cliente (**domus**) recebe como argumentos de entrada:

- IP do servidor (--ip);
- Porto do servidor (--port);
- Pedido a efetuar (--request). Este parâmetro pode ter um dos seguintes valores: ***status***, ***on***, ***off***;
- Dispositivo a atuar (--device). Apenas obrigatório se o pedido for do tipo ***on*** ou ***off***.

Para pedidos do tipo "***status***", o cliente deverá escrever o estado da instalação de domótica no seguinte formato: **Dispositivo X: <Estado>**, em que X representa o nome do dispositivo e <Estado> os pares Ligado/Aberto ou Desligado/Fechado consoante o estado do dispositivo. Para os restantes pedidos, o cliente deverá escrever uma mensagem apropriada consoante a resposta do servidor.

Considere os seguintes exemplos:

```
./domus --ip 127.0.0.1 --port 1234 --request status
        Portão da garagem: Desligado/Fechado
        Iluminação do hall de entrada: Ligado/Aberto
        Iluminação sala: Ligado/Aberto
        Iluminação jardim: Desligado/Fechado
        Persiana 1: Ligado/Aberto
        Persiana 2: Desligado/Fechado
        Persiana 3: Ligado/Aberto
        Piso radiante: Ligado/Aberto

./domus --ip 127.0.0.1 --port 1234 --request on --device 1
        Comando "Ligar/Abrir" – "Portão da garagem" executado com sucesso

./domus --ip 127.0.0.1 --port 1234 --request on --device 1
        Comando "Ligar/Abrir" – "Portão da garagem" ignorado

./domus --ip 127.0.0.1 --port 1234 --request off --device 2
        Comando "Desligar/Fechar" – "Iluminação do hall de entrada" executado com sucesso

./domus --ip 127.0.0.1 --port 1234 --request off --device 12
        Dispositivo #12 desconhecido
```

Pergunta 2

(Escreva as suas respostas a esta pergunta no diretório **`"~/Ficha/R_NUMERO/Pergunta2"`**. Deve indicar o seu nome completo e número de estudante IPLeiria no ficheiro **README.txt** a ser criado no diretório)

NOTA 1: Não é permitida a chamada a comandos externos através da função *system* ou de outra com funcionalidade similar.

NOTA 2: A solução deve ser implementada com recurso a última versão do *template* **EmptyProject-Client-Server-Template**.

NOTA 3: Assuma que não existe perda de datagramas entre os programas cliente e servidor.

NOTA 4: Código entregue que **não compile** leva à atribuição da classificação de **0 (zero) valores** à resposta.

Como sabe, por omissão, a função `recvfrom` é bloqueante, o que leva a que o processo/*thread* chamante fique bloqueado até que ocorra a receção de dados. Contudo, em certas situações, a receção de dados poderá nunca ocorrer. Considere-se, por exemplo, uma aplicação cliente que enviou um datagrama UDP com um pedido de serviço a um servidor, chamando de seguida a função `recvfrom` com o intuito de receber a resposta do servidor. Contudo, tanto o pedido do cliente como a resposta do servidor podem perder-se, nunca sendo entregue aos respetivos destinatários. Neste cenário, a aplicação cliente ficará bloqueada indefinidamente. Para evitar o bloqueio sem limite temporal da chamada, a API socket apresenta duas possibilidades: i) configurar o `socket` através da função `setsockopt` estabelecendo um limite máximo de espera em operações de `recv/recvfrom`; ii) efetuar uma chamada `recv/recvfrom` não bloqueante, que retorna imediatamente, indicando se foram ou não recebidos dados.

a) Elabore a aplicação cliente UDP `clnt_ping` que se limita a enviar a *string* "`PING`" para um servidor, aguardando de seguida pela receção da string "PONG" por parte do servidor. O endereço IP do servidor é indicado através da opção `--ip <IPv4>/-i <IPv4>`. Por sua vez, o porto do serviço é indicado através da opção `--port<int>/-p<int>`. Lance a aplicação cliente, sem que nenhuma aplicação servidora esteja a executar. O que é que sucede?

b) Com base na aplicação cliente `clnt_ping`, crie a aplicação `clnt_ping_setsockopt` de modo a que uma espera na chamada bloqueante `recvfrom` nunca ultrapasse o número de segundos indicados através da opção `--timeout<seconds>/-t <seconds>`. **Sugestão:** uso da função `setsockopt` com o `level SOL_SOCKET` e a opção `SO_RCVTIMEO`. A estrutura `struct timeval` tem a seguinte definição:

```
#include <sys/time.h>

struct timeval {
      time_t      tv_sec;     /* seconds */
      suseconds_t tv_usec;    /* microseconds */
  };
```

Execute a aplicação clnt_ping_setsockopt sem que a aplicação servidora esteja a executar. O que sucede?

c) Com base na aplicação cliente `clnt_ping`, crie a aplicação `clnt_ping_nonblocking`. Esta aplicação deve fazer uso da chamada `recvfrom` em modo não bloqueante. Para o efeito deve ser especificado `MSG_DONTWAIT` como `flag` e monitorizar o valor de `errno` quando a função `recvfrom` devolve `-1`. A aplicação deve proceder a um máximo de três tentativas para receção no `socket` UDP, sendo cada tentativa intervalada por um segundo de espera.

Pergunta 3

(Escreva as suas respostas a esta pergunta no diretório **`"~/Ficha/R_NUMERO/Pergunta3"`**. Deve indicar o seu nome completo e número de estudante IPLeiria no ficheiro **README.txt** a ser criado no diretório)

NOTA 1: Não é permitida a chamada a comandos externos através da função *system* ou de outra com funcionalidade similar.

NOTA 2: A solução deve ser implementada com recurso a última versão do template **`EmptyProject-Client-Server-Template`**.

NOTA 3: Código entregue que **não compile** leva à atribuição da classificação de **0 (zero) valores** à resposta.

Pretende-se que elabore, recorrendo à linguagem C, o sistema cliente/servidor **euroTostoes**. O sistema **euroTostoes** é composto por duas aplicações: **euroTostoes_svc** (aplicação servidor) e **euroTostoes_clnt** (aplicação cliente), que comunicam através do protocolo de transporte **UDP**/**IPv4**. O objetivo é conseguir obter uma chave aleatória de números inteiros que possa ser jogada no Euromilhões. Deste modo, a chave deve ser composta por **cinco** números de 1 a 50, e ainda **dois** números de 1 a 12 ("estrelas").

A aplicação euroTostoes_svc

Sempre que recebe um datagrama com a letra N, devolve um número inteiro aleatório entre 1 e 50, e sempre que recebe um datagrama com a letra E, devolve um número inteiro aleatório entre 1 e 12. Qualquer outro tipo de datagrama é ignorado. A aplicação **euroTostoe**s_**svc** deve ser iniciada com os seguintes argumentos:

- -p, --port <INT>: porto UDP **[opção obrigatória]**
- -s, --seed <INT>: número inteiro empregue para inicializar o gerador de números aleatórios **[opção não obrigatória]**

A opção "-s/--seed" é empregue para inicializar o gerador de números aleatórios com um número inteiro, que deve estar entre 0 e 2^{16}-1. Caso a opção "-s/--seed" não seja especificada (é uma opção não obrigatória), o gerador de números aleatórios deve ser inicializado com o valor 1.

A aplicação euroTostoes_clnt

Deve elaborar uma potencial chave do Euromilhões, isto é, cinco números inteiros distintos entre 1 e 50, mais duas estrelas (distintas) que correspondem a números inteiros entre 1 e 12. Para o efeito, a aplicação cliente deve solicitar cada um dos números ao servidor **euroTostoes_svc**. A aplicação **euroTostoes_clnt** deve filtrar as respostas do servidor que correspondam a números repetidos, solicitando, sempre que necessário, um novo número à aplicação **euroTostoes_svc**. As opções da linha de comando para **euroTostoes_clnt** são as seguintes:

- -p, --port <INT>: porto UDP do servidor a contactar **[opção obrigatória]**
- -i, --ip <IPv4>: endereço IPV4 (formato *dotted-decimal*) da aplicação servidor a contactar **[opção obrigatória]**

Um exemplo de interação entre as aplicações cliente/servidor é mostrado de seguida:

`./euroTostoes_svc --port 4444 -s 37`	**`./euroTostoes_clnt --ip 127.0.0.1 --port 4444`**
`[SVC] waiting for UDP queries / port 4444 /seed 37`	
	`[CLNT] sending 'N' request`
`[SVC] 'N' received: sending 27`	

	[CLNT] received 27 – NUMBERS: [27] [CLNT] sending ‘N’ request
[SVC] ‘N’ received: sending 32	
	[CLNT] received 32 – NUMBERS: [27 32] [CLNT] sending ‘N’ request
[SVC] ‘N’ received: sending 21	
	[CLNT] received 21 – NUMBERS: [27 32 21] [CLNT] sending ‘N’ request
[SVC] ‘N’ received: sending 32	
	[CLNT] received 32 –duplicated – discarding it [CLNT] sending ‘N’ request
[SVC] ‘N’ received: sending 19	
	[CLNT] received 19 –NUMBERS: [27 32 21 19] [CLNT] sending ‘N’ request
[SVC] ‘N’ received: sending 45	
	[CLNT] received 45 – NUMBERS:[27 32 21 19 45] [CLNT] got 5 numbers – entering stars mode [CLNT] sending ‘E’ request
[SVC] ‘E’ received: sending 12	
	[CLNT] received 12 – STARS:[12] [CLNT] sending ‘E’ request
[SVC] ‘E’ received: sending 10	
	[CLNT] received 10 – OK STARS:[12,10] [CLNT] FINISHED [CLNT] NUMBERS:[19,21,27,32,45] (sorted) [CLNT] STARS: [10,12] (sorted)

Pergunta 4

(Escreva as suas respostas a esta pergunta no diretório **`"~/Ficha/R_NUMERO/Pergunta4"`**. Deve indicar o seu nome completo e número de estudante IPLeiria no ficheiro **README.txt** a ser criado no diretório)

NOTA 1: Não é permitida a chamada a comandos externos através da função *system* ou de outra com funcionalidade similar.

NOTA 2: A solução deve ser implementada com recurso à ultima versão do *template* **EmptyProject-Client-Server-Template**.

NOTA 3: Código entregue que **não compile** leva à atribuição da classificação de **0 (zero) valores** à resposta.

Pretende-se que implemente, recorrendo à linguagem C, um programa cliente/servidor elementar que permita efetuar o registo de novos utilizadores num serviço. Os programas cliente e servidor devem comunicar através do protocolo de transporte TCP.

O servidor

Recebe como argumentos da linha de comandos:

- Porto (--porto ou -p), a partir do qual vai receber os dados dos utilizadores para registo;
- Número máximo de registos que o serviço aceita (--max_users ou -u).

O servidor deve ficar à espera de clientes, até atingir o número máximo de utilizadores indicado na linha de comando. Por sua vez, o cliente efetua um pedido de registo de utilizador, enviando ao servidor, o nome e o telefone. Um nome tem no máximo 74 caracteres, enquanto o número de telefone é representado por uma *string*, que tem um máximo de 14 caracteres (e.g., "+351.123456789").

Recebido o pedido do cliente, o servidor deve atribuir um *userid* (representado por um inteiro sem sinal de 4 *bytes*), mantendo ainda numa apropriada estrutura de dados, os dados *userid*, nome e telefone. O *userid* atribuído a esse utilizador deve ser incremental, sendo enviado ao cliente após a receção e armazenamento do nome e telefone do novo utilizador.

Após o envio do *userid* ao programa cliente, o programa servidor encerra a ligação com o programa cliente, colocando-se de novo à escuta para registos de clientes. Caso o número máximo de registos de utilizadores tenha sido alcançado, o programa servidor termina e mostra, na saída padrão, a mensagem "[servidor]: max. numero de registos (N) – terminando", em que N corresponde ao número máximo de registos especificados pela opção "-u /--max_users".

O cliente

Deve receber como argumentos da linha de comandos:

- IP do servidor (--ip ou -i);
- Porto do servidor (--porto ou -p) a utilizar para fazer registo de um novo utilizador.

Depois de solicitar e validar a introdução do nome e do telefone por parte do utilizador, a aplicação cliente deve fazer a ligação ao servidor e, de seguida, enviar os dados recolhidos anteriormente. Após o envio dos dados do utilizador, o cliente deverá esperar a receção do *userid* atribuído pelo servidor e mostrá-lo no ecrã ao utilizador, terminando de seguida.

Considere os seguintes exemplos:

Servidor:

```
$ ./servidor -p 1234 -u 12
Servidor: TCP/1234
A espera de pedidos de registo de utilizadores...
Recebido pedido de registo de utilizador...
Utilizador registado:
Userid-> 1
Nome-> Programacao Avancada
Telefone-> 244888888

A espera de pedidos de registo de utilizadores...
Recebido pedido de registo de utilizador...
Utilizador registado:
Userid-> 2
Nome-> Exame Pratico 2
Telefone-> +351991122334

A espera de pedidos de registo de utilizadores...
```

Cliente:

```
$ ./cliente --ip 127.0.0.1 --porto 1234
Indique o seu nome: Programacao Avancada
Indique o seu telefone: 244888888
A efetuar o registo...
Ficou registado com o USERID => 1
```

Pergunta 5

(Escreva as suas respostas a esta pergunta no diretório **`"~/Prova02/R_NUMERO/Pergunta"`**. Deve indicar o seu nome completo e número de estudante IPLeiria no ficheiro **README.txt** a ser criado no diretório)

NOTA 1: não é permitida a chamada a comandos externos através da função *system* ou de outra com funcionalidade similar.

NOTA 2: a solução deve ser implementada com recurso aos ficheiros do diretório **EmptyProject-client-server-template.v2.02.zip**

NOTA 3: código entregue que **não compile** através do utilitário *make* e do respetivo *makefile* leva à atribuição da classificação de **0 (zero) valores** à resposta.

Pretende-se que elabore a aplicação cliente/servidor TCP concorrente `cipher` cujo propósito é o de cifrar/decifrar uma mensagem utilizando o operador XOR. Para o efeito, a aplicação cliente **cipher_clnt** recebe, através da opção de linha de comandos `--message <message>`, uma mensagem a cifrar/decifrar e opcionalmente o parâmetro `--key <key>`. O parâmetro `--key <key>` é um inteiro de 8 bits (uint8_t) que é empregue como chave na operação de cifragem.

O comportamento do serviço `cipher` é o seguinte: **i)** Caso o parâmetro `--key` **não** seja fornecido, o cliente deve solicitar ao servidor que cifre a mensagem (pedido CIFRAR). O servidor responderá enviando duas mensagens TCP: uma que contém uma chave criada aleatoriamente, seguido de outra que contém a mensagem cifrada; **ii)** Caso o parâmetro `--key` **seja fornecido**, o cliente deve solicitar ao servidor a decifragem da mensagem (pedido DECIFRAR). Para o efeito, o cliente envia ao servidor a chave indicada na linha de comando. O servidor irá responder com a mensagem decifrada.

A aplicação **`cipher_svr`** desempenha o papel de servidor. Esta aplicação aguarda por pedidos de serviço, sendo que quando recebe um pedido deve proceder consoante o tipo de pedido (**sugestão:** uint8_t).

Se o pedido enviado pelo cliente for do tipo **CIFRAR**, o servidor deve:

- Gerar um número aleatório entre 1 e 127 (**uint8_t**);
- Efetuar a receção da mensagem a cifrar (assuma um máximo de 1024 caracteres);
- Substituir cada caracter da mensagem pela operação: **`mensagem[i] XOR chave;`**
- Caso o resultado da operação de cifragem de um caracter seja inferior ao valor **32**, deverá ser mantido o caracter original[1];
- Após efetuar a cifragem, o servidor deve enviar ao cliente, em duas mensagens separadas, a chave, seguida da mensagem cifrada, fechando de seguida a ligação.

Se o pedido for de o tipo **DECIFRAR**, o servidor deve:

- Efetuar a receção da chave;
- Efetuar a receção da mensagem (assuma um máximo de 1024 caracteres);
- Reverter o algoritmo de cifragem, substituindo cada caracter da mensagem com valor >= **32** pela operação XOR com a chave;
- Depois de efetuar a conversão, o servidor deve enviar ao cliente a mensagem decifrada e terminar a ligação.

Tenha em atenção que **`cipher_svr`** é um servidor concorrente, pelo que deve ser capaz de atender vários clientes simultaneamente.

Os parâmetros da linha de comando da aplicação cliente **`cipher_clnt`** são:
`--ip/-i <IPv4_address>`: endereço IPv4 do servidor. Caso não seja especificado, deve ser assumido o endereço "127.0.0.1".
`--port/-p <int>`: porto remoto (TCP) do servidor. **Este parâmetro é obrigatório**.
`--message/-m <message>`: mensagem a cifrar/decifrar. **Este parâmetro é obrigatório**.
`--key/-k <key>`: chave a utilizar na operação de decifrar. **Parâmetro não obrigatório**, sendo que, caso não seja especificado, a operação a realizar é CIFRAR.

Por sua vez, o servidor **`cipher_svr`** recorre ao seguinte parâmetro da linha de comando:

[1] Um valor inferior a 32 corresponde a um caracter dito não "*printável*".

`--port/-p <int>`: porto de escuta TCP da aplicação. **Este parâmetro é obrigatório**.

Ambas as aplicações devem validar os parâmetros, nomeadamente o porto (valor entre 1 e 65535), e o endereço IP (aplicação cliente). No caso de algum parâmetro estar incorreto, deve ser escrito no canal de erro padrão uma apropriada mensagem de erro e, seguidamente, terminar a aplicação.

Considere os seguintes exemplos de execução:

Exemplo 1 (CIFRAR)

```
./cipher_clnt --ip 127.0.0.1 --port 1234 -m 'Hello World'
>> KEY: 3
>> MESSAGE: Kfool#Tlqog
```

Exemplo 2 (DECIFRAR)

```
./cipher_clnt --ip 127.0.0.1 --port 1234 -m 'Kfool#Tlqog' -k 3
>> MESSAGE: Hello World
```