

## Ficha 3 – Programação com *sockets* (TCP e UDP)

Ano Letivo de 2023/2024

---

### 1 – Introdução

Nesta ficha iremos começar a abordar a construção de aplicações que fazem uso dos serviços de comunicação da camada de transporte, em particular dos protocolos TCP e UDP. Para o desenvolvimento das suas aplicações deverá utilizar a Virtual Machine fornecida para apoio às aulas PL.

#### Avaliação da Ficha

- Esta Ficha vale **1 valor** (em 20).
- Deverá submeter as suas respostas aos Exercícios 2 e 4 via Moodle, até ao dia 17/mar.

### 2 – Programação com *sockets* (TCP)

A programação com *sockets* surgiu no sistema operativo BSD Unix 4.1c, em 1980, e representa na atualidade o modelo de programação de rede utilizado em virtualmente todos os sistemas operativos (Windows, Unix e macOS, entre outros). Existem dois tipos principais de *sockets*: os Unix *sockets* (FIFOs/Pipes no sistema de ficheiros) e os *sockets* Internet, sendo neste últimos que iremos focar a nossa atenção nesta ficha prática.

O Protocolo TCP (Transmission Control Protocol) é um dos protocolos principais da Internet, garantindo a entrega ordenada e livre de erros dos pacotes transmitidos. Tal como o UDP, o TCP atua na camada de transporte e recorre aos serviços do Protocolo IP (Internet Protocol) na camada de rede para transmissão da informação. A figura seguinte apresenta um modelo genérico de interação (comunicação) entre um cliente e um servidor com TCP:

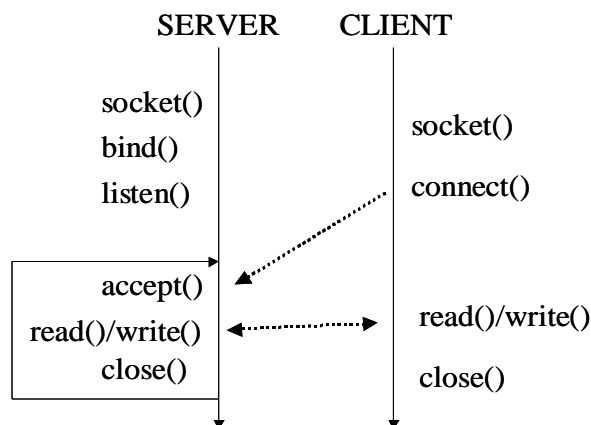


Figura 1 - Utilização de *sockets* na comunicação com TCP

## Descrição das funções necessárias

```
#include <sys/types.h>
#include <sys/socket.h>

/* Cria um novo socket. Ver página de manual no Linux: "man socket" */
int socket(int domain, int type, int protocol)

domain:      Domínio no qual o socket será usado
              (processos Unix / internet)
              (AF_UNIX, AF_INET, AF_INET6, ...)
type:        Tipo de ligação (orientada a ligações ou datagrama)
              (SOCK_STREAM, SOCK_DGRAM, ...)
protocol:    Forma de comunicação (0 para protocolo por omissão,
              IPPROTO_TCP para TCP, IPPROTO_UDP para UDP)

Protocolos por default:
Domínio AF_INET e tipo SOCK_STREAM: TCP
Domínio AF_INET e tipo SOCK_DGRAM: UDP

DEVOLVE:     "Descritor de socket"
```

```
#include <sys/types.h>
#include <sys/socket.h>

/* Associa um socket a um determinado endereço. Página de manual no Linux:
"man 2 bind" */
int bind(int fd, const struct sockaddr *address, socklen_t address_len)

fd:          "Descritor de socket"
address:      Ponteiro para o endereço a associar ao socket
address_len:  Dimensão da estrutura de dados indicada em <address>

DEVOLVE:     0 para sucesso, -1 para erro

Internet Sockets:

struct sockaddr_in {
    short      sin_family;    // AF_INET
    u_short    sin_port;     // porto a associar
    struct in_addr sin_addr;  // INADDR_ANY = qualquer
                                // endereço do host
    char       sin_zero;     // padding, deixar em branco
}
```

```

}

struct in_addr {
    unsigned long s_addr;
};

```

**Nota:**

2) Endereços especiais:

**INADDR\_ANY** - (0.0.0.0) - quando especificado na função *bind*, o socket ficará ligado a todas as interfaces locais da máquina

**INADDR\_LOOPBACK** - (127.0.0.1) - refere sempre o localhost via o interface de loopback

```

#include <sys/types.h>
#include <sys/socket.h>

/* Aguardar pela recepção de ligações. Ver página de manual no Linux: "man
listen" */

```

```

int listen(int fd, int backlog)

```

fd: "Descritor de *socket*"  
backlog: Quantos clientes são mantidos em espera (a aguardar o *accept*) antes de haver recusa de ligação (com a mensagem "Connection Refused")

DEVOLVE: 0 para sucesso, -1 para erro

```

#include <sys/types.h>
#include <sys/socket.h>

/* Aceita uma ligação. Ver página de manual: "man 2 accept" */

```

```

int accept(int fd, const struct sockaddr *address,
socklen_t* address_len)

```

fd: "Descritor de *socket*"  
address: Estrutura de dados que vai ser preenchida com informação sobre a ligação que está a ser estabelecida  
address\_len: Comprimento do *buffer* <address>. No final da chamada irá conter o tamanho (em octetos) da estrutura <address>

DEVOLVE: "Descritor de *socket*" da ligação aceite, -1 em caso de erro

```

#include <sys/types.h>
#include <sys/socket.h>

/* Inicia uma ligação num socket. Ver página de manual: "man connect" */

```

```

int connect(int fd, const struct sockaddr *address,
socklen_t address_len)

```

fd: "Descritor de *socket*"  
address: Endereço do servidor ao qual se pretende ligar  
address\_len: Dimensão da estrutura <address>

DEVOLVE: 0 para ligação estabelecida, -1 no caso contrário

## Descrição de funções auxiliares

Consultar mais detalhes na página do manual do Linux: "man <nome\_função>".

```
#include <sys/types.h>
#include <sys/socket.h>

/* Converte nome para endereço. Pág. de manual no Linux: "man gethostbyname" */
struct hostent * gethostbyname(const char* name)

DEVOLVE:      Estrutura com o endereço Internet correspondente ao nome

Estrutura hostent:
O mapeamento entre o nome do host e o endereço é representado pela estrutura
struct hostent:

struct hostent {
    char    *h_name;           // nome oficial do host
    char    **h_aliases;       // lista de aliases
    int     h_addrtype;        // hostaddrtype (ex.: AF_INET6)
    int     h_length;          // comprimento do endereço
    char    **h_addr_list;     // lista de end. terminada com null
};

/* 1st addr, net byte order */
#define h_addr h_addr_list[0]

The h_addr definition is for backward compatibility, and is the first address
in the list of addresses in the hostent structure.
```

### Nota:

Na arquitetura i80x86 a ordem dos bytes é a *little-endian* (primeiro é armazenado na memória o byte menos significativo), enquanto as comunicações em rede utilizam (enviam) primeiro os bytes mais significativos (*big-endian*).

```
#include <arpa/inet.h> ou <netinet/in.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

**htonl()** - (*host to network long*) converte um inteiro sem sinal da ordem de bytes do *host* para a ordem de bytes da rede.

**htons()** - (*host to network short*) converte um inteiro *short* sem sinal da ordem de bytes do *host* para a ordem de bytes da rede.

**ntohl()** - (*network to host long*) converte um inteiro sem sinal da ordem de bytes da rede para a ordem de bytes do *host*.

**ntohs()** - (*network to host short*) converte um inteiro *short* sem sinal *netshort* da ordem de bytes da rede para a ordem de bytes do *host*.

```

#include <arpa/inet.h>

/* Converte um endereço IPv4 para uma string com o formato xxx.xxx.xxx.xxx =>
n(network) to p(presentation) */
const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);

af:    família AF_INET (para IPv4) ou AF_INET6 (para IPv6)
src:   ponteiro para uma estrutura struct in_addr ou struct in6_addr
dst:   string destino
size:  tamanho máximo da string destino (o comprimento máximo é de
      INET_ADDRSTRLEN e INET6_ADDRSTRLEN)

/* Converte uma string com o endereço IP num valor para a estrutura struct
in_addr ou struct in6_addr => p(presentation) to n(network) */
int inet_pton(int af, const char *src, void *dst);

```

## Exemplo TCP (servidor e cliente) (código fonte incluído nos materiais da Ficha)

```
/*
*****
* SERVIDOR no porto 9000, à escuta de novos clientes. Quando surgem
* novos clientes os dados por eles enviados são lidos e descarregados no ecran.
*****
*/
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <netdb.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>

#define SERVER_PORT 9000
#define BUF_SIZE 1024

void process_client(int fd);
void erro(char *msg);

int main() {
    int fd, client;
    struct sockaddr_in addr, client_addr;
    int client_addr_size;

    bzero((void *) &addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(SERVER_PORT);

    if ( (fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) erro("na funcao socket");
    if ( bind(fd, (struct sockaddr*)&addr, sizeof(addr)) < 0) erro("na funcao bind");
    if( listen(fd, 5) < 0) erro("na funcao listen");
    client_addr_size = sizeof(client_addr);

    while (1) {
        //clean finished child processes, avoiding zombies
        //must use WNOHANG or would block whenever a child process was still working
        while(waitpid(-1, NULL, WNOHANG)>0);
        // wait for new connection
        client = accept(fd, (struct sockaddr *)&client_addr, (socklen_t *)&client_addr_size);
        if (client > 0) {
            if (fork() == 0) {
                close(fd);
                process_client(client);
                exit(0);
            }
            close(client);
        }
    }
    return 0;
}

void process_client(int client_fd){
    int nread = 0;
    char buffer[BUF_SIZE];

    do {
        nread = read(client_fd, buffer, BUF_SIZE-1);
        buffer[nread] = '\0';
        printf("%s", buffer);
        fflush(stdout);
    } while (nread > 0);
    close(client_fd);
}

void erro(char *msg){
    printf("Erro: %s\n", msg);
    exit(-1);
}
```

```

/*****
* CLIENTE liga ao servidor (definido em argv[1]) no porto especificado
* (em argv[2]), escrevendo a palavra predefinida (em argv[3]).
* USO: >cliente <enderecoServidor> <porto> <Palavra>
*****/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>

void erro(char *msg);

int main(int argc, char *argv[]) {
    char endServer[100];
    int fd;
    struct sockaddr_in addr;
    struct hostent *hostPtr;

    if (argc != 4) {
        printf("cliente <host> <port> <string>\n");
        exit(-1);
    }

    strcpy(endServer, argv[1]);
    if ((hostPtr = gethostbyname(endServer)) == 0)
        erro("Não consegui obter endereço");

    bzero((void *) &addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = ((struct in_addr *) (hostPtr->h_addr))->s_addr;
    addr.sin_port = htons((short) atoi(argv[2]));

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        erro("socket");
    if (connect(fd, (struct sockaddr *) &addr, sizeof(addr)) < 0)
        erro("Connect");
    write(fd, argv[3], 1 + strlen(argv[3]));
    close(fd);
    exit(0);
}

void erro(char *msg) {
    printf("Erro: %s\n", msg);
    exit(-1);
}

```

### Notas:

- Um endereço associado a um *socket* TCP fica indisponível durante algum tempo após o *socket* ser fechado. Isto só não acontece se usarmos a *flag* `SO_REUSEADDR` (ver a função `setsockopt()`). Esta *flag* permite que vários *sockets* possam estar associados a um mesmo endereço. Não é aconselhado o uso desta *flag*, exceto em casos especiais.
- Devem sempre fechar o *socket* antes de terminar o programa.
- Uma solução alternativa em caso de erro no *bind*, é mudar o número do porto em utilização no servidor.

## Exercícios de programação com sockets TCP:

### Exercício 1: (exercício não avaliado)

Modifique as aplicações cliente e servidor apresentadas, de modo a obter as seguintes funcionalidades:

- O servidor escreve na consola o endereço IP e o porto do cliente que lhe está a ligar; dá também um número a cada cliente novo que liga;
- O servidor devolve ao cliente uma mensagem de texto com o endereço IPv4, o porto do qual o cliente está a ligar e o número de clientes que já estabeleceram ligação.

Exemplo:

#### Servidor

```
user@user-virtualbox$ ./servidor
** New message received **
Client 1 connecting from (IP:port) 127.0.0.1:58511 says "Bom dia!"
** New message received **
Client 2 connecting from (IP:port) 127.0.0.1:59023 says "Olá!"
```

#### Cliente:

```
user@user-virtualbox$ ./cliente 127.0.0.1 9000 "Bom dia!"
Received from server:
Server received connection from (IP:port) 127.0.0.1:58511; already received 1 connections!

user@user-virtualbox$ ./cliente 127.0.0.1 9000 "Olá!"
Received from server:
Server received connection from (IP:port) 127.0.0.1:59023; already received 2 connections!
```

### Exercício 2:

Com este exercício pretende-se simular o processo de obtenção, por parte de um cliente, do endereço IP registado para um determinado nome de domínio.

Modifique as aplicações cliente e servidor fornecidas, de modo a construir um servidor e cliente com as funcionalidades descritas de seguida.

Devera ter um servidor e admitir ligações de vários clientes simultaneamente. Quando um cliente inicia a ligação, o servidor envia ao cliente a mensagem seguinte:

Bem-vindo ao servidor de nomes do DEI. Indique o nome de domínio

O cliente envia um nome de domínio com o formato seguinte: www.dei.uc.pt

De seguida, o servidor deve procurar o nome de domínio num ficheiro de texto e responder ao cliente com o endereço IP associado, com uma mensagem como a seguinte:

O nome de domínio www.dei.uc.pt tem associado o endereço IP 193.137.203.227

Caso o nome não seja encontrado no ficheiro, o servidor deverá responder da seguinte forma:

O nome de domínio www.dei.uc.pt não tem um endereço IP associado

O cliente pode continuar a enviar nomes de domínio e o servidor devera continuar a responder. Quando o cliente quiser fechar a sessão, devera enviar a mensagem seguinte:

SAIR



Quando o servidor recebe a mensagem “SAIR”, o servidor devera responder com a mensagem seguinte:

|           |
|-----------|
| Até logo! |
|-----------|

O servidor deverá poder aceitar ligações (pedidos) de vários clientes em simultâneo, e terminar a sua execução com Ctrl+C.

O formato do ficheiro de texto deve ser como de seguida:

|   |
|---|
| www.dei.uc.pt 193.137.203.227<br>microsoft.com 20.112.52.29<br>meo.com 3.33.139.32<br>autenticacao.gov.pt 62.28.186.215 |
|---|

## Programação com sockets (UDP)

Ao contrário do que acontece com o TCP (*Transmission Control Protocol*), no protocolo UDP (*User Datagram Protocol*) não existem ligações, sendo que consequentemente não é necessário manter informação de estado relativamente a associações entre computadores. Isto significa que um servidor UDP não aceita ligações e, da mesma forma, um cliente UDP não tem a necessidade de estabelecer uma ligação ao servidor. Os pacotes UDP são enviados isoladamente entre sistemas, sem quaisquer garantias em relação à sua entrega ou à sua ordenação na chegada ao sistema de destino.

### Descrição das funções principais

- Na criação do *socket* o tipo (*socket\_type*) deve indicar a utilização de *datagrams* em vez de *data streams*:

```
#include <sys/types.h>
#include <sys/socket.h>

/* Cria um novo socket */
int socket(int domain, int type, int protocol)

domain:      Domínio no qual o socket será usado
              (processos Unix / internet)
              (AF_UNIX ou AF_INET)

type:        Tipo de ligação (orientada a ligações ou utilizando datagramas,
              SOCK_STREAM ou SOCK_DGRAM)

protocol:    Forma de comunicação (0 para protocolo por omissão)

              Protocolo por default:
              Domínio AF_INET e tipo SOCK_DGRAM: UDP

DEVOLVE:     Descritor de socket
```

- Para além da criação do *socket* propriamente dito, é necessário utilizar a função *bind* no servidor para definir a porta a utilizar, após o que o servidor pode receber pacotes UDP de vários clientes.
- Um programa pode utilizar as funções *sendto* e *recvfrom* (entre outras) para enviar ou receber pacotes UDP de outro computador. Estas funções recebem ou devolvem o endereço e porto do outro computador:

```
#include <sys/types.h>
#include <sys/socket.h>

/* Recebe uma mensagem através de um socket */
int recvfrom(int sockfd, void *buf, int len, int flags,
             struct sockaddr *src_addr, socklen_t *addrlen)

sockfd:      socket onde é recebida a mensagem.
buf:         buffer para armazenamento da mensagem.
len:         número máximo de bytes a ler (de acordo com o tamanho do buffer).
flags:       controlo da operação de leitura (utilizar comando "man recvfrom"
              no linux para mais informação).
src_addr:    estrutura para armazenamento do endereço de origem da mensagem.
addrlen:     tamanho do endereço de origem da mensagem.

A função devolve: em caso de sucesso o número de bytes (tamanho da mensagem
UDP) recebidos.
```

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
/* Envia uma mensagem */
```

```
int sendto(int sockfd, void *buf, int len, int flags,
           struct sockaddr *dest_addr, socklen_t addrlen)
```

Parâmetros:                    semelhantes aos da função anterior (ver página de manual com o comando "man sendto" no Linux).

A função devolve: em caso de sucesso o número de bytes enviados.

### **Exemplo (implementação servidor UDP)** (código fonte incluído nos materiais da Ficha)

Apresenta-se a seguir um exemplo de um programa servidor que recebe mensagens UDP:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

#define BUFLen 512      // Tamanho do buffer
#define PORT 9876      // Porto para recepção das mensagens

void erro(char *s) {
    perror(s);
    exit(1);
}

int main(void) {
    struct sockaddr_in si_minha, si_outra;

    int s, rcv_len;
    socklen_t slen = sizeof(si_outra);
    char buf[BUFLen];

    // Cria um socket para recepção de pacotes UDP
    if((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1){
        erro("Erro na criação do socket");
    }

    // Preenchimento da socket address structure
    si_minha.sin_family = AF_INET;
    si_minha.sin_port = htons(PORT);
    si_minha.sin_addr.s_addr = htonl(INADDR_ANY);

    // Associa o socket à informação de endereço
    if(bind(s, (struct sockaddr*)&si_minha, sizeof(si_minha)) == -1){
        erro("Erro no bind");
    }

    // Espera recepção de mensagem (a chamada é bloqueante)
    if((rcv_len=recvfrom(s, buf, BUFLen, 0, (struct sockaddr *)&si_outra,
        (socklen_t *)&slen)) == -1){
        erro("Erro no recvfrom");
    }

    // Para ignorar o restante conteúdo (anterior do buffer)
    buf[rcv_len]='\0';

    // Envia para a consola a mensagem recebida
    printf("Recebi uma mensagem do sistema com o endereço %s e o porto %d\n",
        inet_ntoa(si_outra.sin_addr), ntohs(si_outra.sin_port));
    printf("Conteúdo da mensagem: %s\n", buf);

    // Fecha socket e termina programa
    close(s);
    return 0;
}
```

## Exercícios (programação com sockets UDP e Wireshark):

### Exercício 3: (exercício não avaliado):

Utilize a aplicação do exemplo anterior e o programa “netcat” como cliente para envio de mensagens UDP. Valide o envio da mensagem pelo cliente e a sua correta recepção no servidor.

Síntaxe de utilização (no Linux):

```
nc <-u> <Endereço IP servidor> <Porto>
```

Exemplo:

#### Servidor

```
user@user-virtualbox$ ./servidor
Recebi uma mensagem do sistema com o endereço 127.0.0.1 e o porto 42806
Conteúdo da mensagem: Bom dia!
```

#### Cliente:

```
user@user-virtualbox$ nc -u localhost 9876
Bom dia!
```

### Exercício 4:

Utilizando o servidor do exemplo anterior, altere-a na medida do necessário, e desenvolva o cliente associado, para criar um conversor de base de números. O cliente envia um número em base decimal e o servidor responde com a conversão a base binária e hexadecimal.

Exemplo:

O Cliente envia:

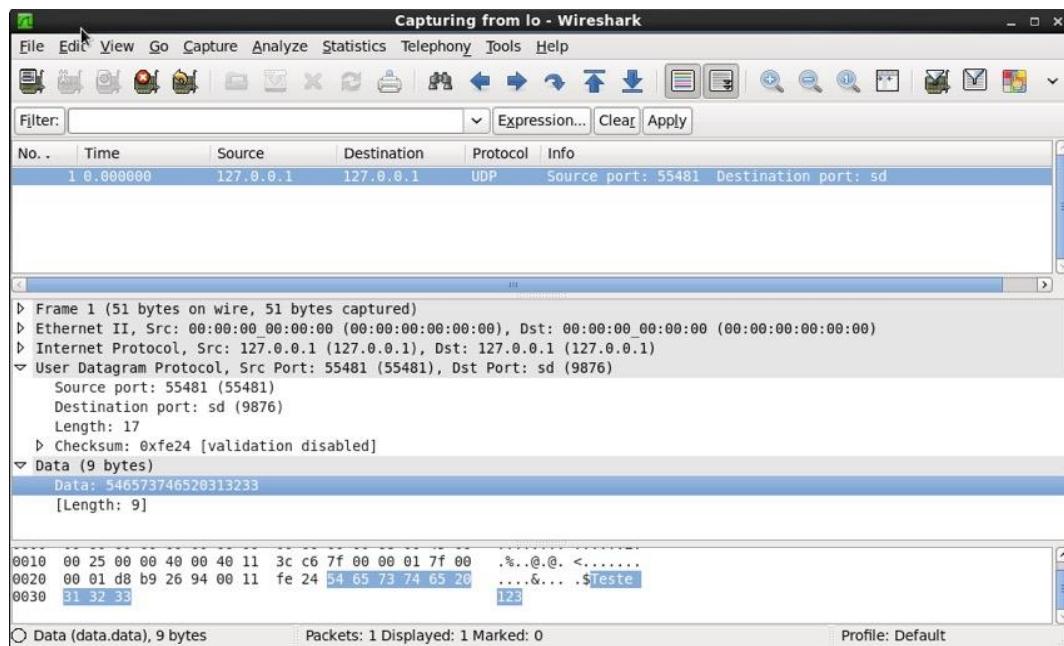
```
25
```

O Servidor responde:

```
Número em binário: 0001 1001
Número em hexadecimal: 0x19
```

### Exercício 5:

Utilizando um *network sniffer* como o *wireshark* (<http://www.wireshark.org/>) observe as mensagens UDP e TCP trocadas entre o cliente e o servidor, bem como o seu conteúdo e restante informação. A fig. seguinte apresenta um exemplo de utilização do *wireshark* com esse propósito.



#### Notas:

- Caso a aplicação *wireshark* não esteja instalada poderá instalar o respetivo package recorrendo ao comando seguinte: “`sudo apt-get install wireshark`”.
- Para executar (chamar) o *wireshark* executar o comando “`sudo wireshark`” (o `sudo` executa o programa em modo de administrador, por forma a garantir que tem acesso a todas as interfaces de rede).