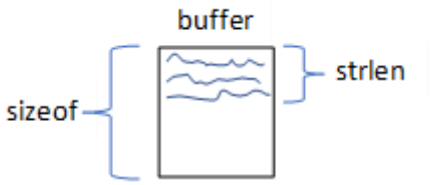
Conceitos e processos básicos



sizeof() → retorna o tamanho total do buffer

strlen() → retorna o comprimento de um array de caracteres

atoi() → converte um array de caracteres (string) num inteiro

strcmp() → compara 2 arrays de caracteres (strings). Retorna 0 (zero) se forem iguais

---

argc → indica o número de argumentos que foram passados na linha de comandos

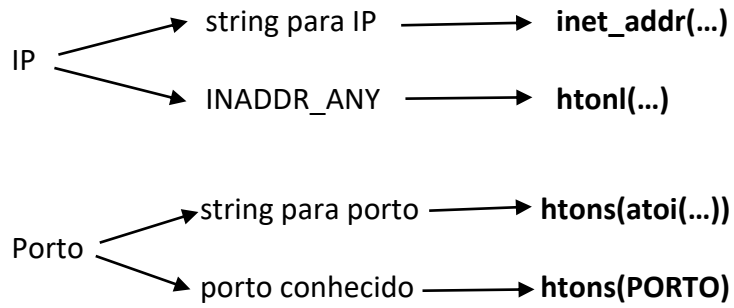
argv → array de argumentos que foram passados na linha de comandos

Exemplo:

- ./Cliente.exe "Ola Mundo" **argc** = 2

  argv[0] **argv[1]**

Registar:



Mostrar

IP ⟶ **inet_ntoa(…)**

Porto ⟶ **ntohs(…)**

inet_addr() → converte uma string no formato *dotted decimal* (exemplo: 127.0.0.1) para um endereço IP que será reconhecido pela rede TCP/IP.

htonl() → host to network long. De modo básico, estamos a converter um endereço de host para algo que será reconhecido pela rede TCP/IP.

htons() → host to network short. De modo básico, estamos a converter um número em algo que será reconhecido pela rede TCP/IP.

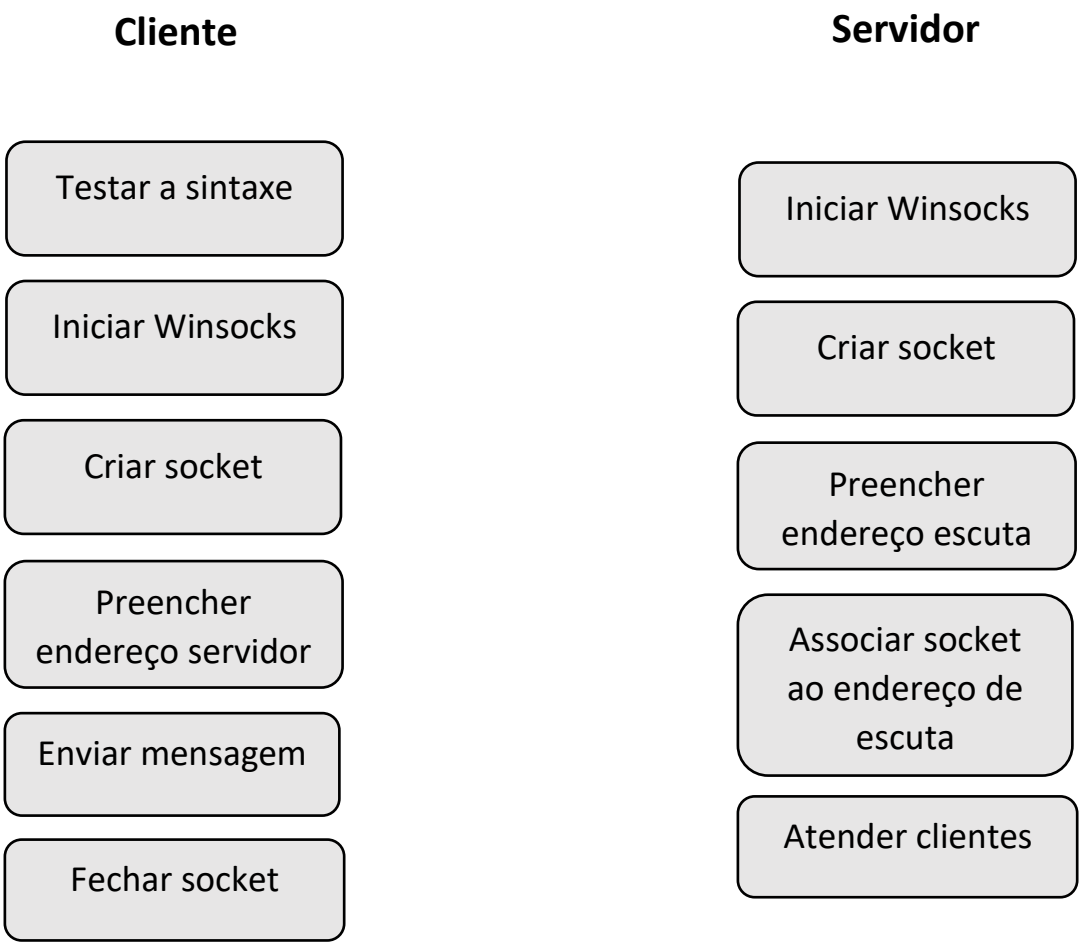
ntohs() → network to host short. De modo básico, estamos a converter um porto reconhecido pela rede TCP/IP para um número.

inet_ntoa() → converte um endereço IP para algo que será reconhecido por nós (string no formato dotted decimal)

UDP

- Orientado a datagramas
- Não é fiável, ou seja, não garante a entrega/recepção dos datagramas
- Não é orientado a ligação (tipo **carta correio simples**, ou seja, sempre que queremos enviar um datagrama temos que especificar o endereço de destino)

Passos básicos do cliente e servidor



Criar socket

```
                    1          2          3
sockfd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

1. Especificação da família de endereços. No nosso caso vamos usar endereços no formato IPv4, designados por **PF_INET** (ou AF_INET).

2. Tipo do socket, neste caso socket que suporta datagramas.

3. Protocolo a ser usado, neste caso UDP.
   Nota: em alguns exercícios podem ver este parâmetro com o valor 0 (zero). A função socket permite que este valor seja 0, e nesse caso, não estamos a especificar diretamente que queremos usar o protocolo UDP. O protocolo a usar será definido pelo *service provider*.
   Fonte: https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-socket

---

Preencher endereço

```
1 memset((char*)&serv_addr, 0, sizeof(serv_addr));
2 serv_addr.sin_family = AF_INET;
3 serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR) OU htonl(INADDR_ANY);
4 serv_addr.sin_port = htons(SERV_UDP_PORT);
```

1. Coloca a 0 (zero) todos os bytes da estrutura serv_addr

2. Família de endereços (Internet)

3. Endereço do host
   Cliente: IP do endereço do servidor (usando a função inet_addr)
   Servidor: ao usar o INADDR_ANY (com a função htonl) estamos a dizer que estamos à escuta de datagramas vindos de qualquer endereço de rede (da família especificada no ponto 2)

4. Porto que identifica a aplicação

Nota: Tenham atenção às funções inet_addr(), htonl(), htons(),…

Enviar mensagem

```
           1         2              3           4             5                        6
sendto(sockfd, argv[1], strlen(argv[1]), 0, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

1. Socket (criado anteriormente)

2. Endereço da mensagem que se pretende enviar (neste caso estamos a usar o valor na posição 1 dos argumentos que foram passados na linha de comandos)

3. Comprimento da mensagem que se pretende enviar (**cuidado! strlen()** e não sizeof())

4. Flags (não importante por agora)

5. Endereço da estrutura que contém os dados (família de endereços, IP e Porto) para onde queremos enviar a mensagem.
   Notas:
   a. UDP = carta correio simples, ou seja, têm sempre que indicar o destinatário da mensagem.
   b. Porque temos que fazer o cast?
      i. A função sendTo() está à espera de um ponteiro para uma estrutura do tipo "sockaddr", isto porque existem vários tipos de sockets e esta função está preparada para todos.
      ii. Como nós estamos a usar uma estrutura do tipo "sockaddr_in" - (in -> Internet, ou seja, indicada para comunicações baseadas em IP – Internet Protocol), temos que fazer o cast para a estrutura que a função sendTo() está à espera.

6. Tamanho da estrutura que contém os dados do endereço de destino

Receber mensagem

```
length_addr = sizeof(cli_addr);
recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&cli_addr, &length_addr);
         1       2       3             4     5                          6

buffer[nbytes] = '\0';
        7
```

1. Socket (criado anteriormente)
2. Endereço do array de caracteres onde se pretende guardar a mensagem
3. Tamanho do array de caracteres definido no ponto 2 (**cuidado! sizeof()** e não strlen())
4. Flags (não importante por agora)
5. Endereço da estrutura que será preenchida com os dados (família de endereços, IP e Porto) de onde a mensagem foi recebida
   Nota: ver explicação no ponto 5 da seção "Enviar mensagem"
6. Endereço para o tamanho da estrutura definida no ponto 5
7. Termina a cadeira de caracteres recebidos com "\0"

Associar socket ao endereço de escuta

```
     1                2                3
bind(sockfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

1. Socket (criado anteriormente)
2. Endereço da estrutura que contém os dados do servidor (família de endereços, IP e Porto)
   Nota: ver explicação no ponto 5 da seção "Enviar mensagem"
3. Tamanho da estrutura definida no ponto 2

Obter informação do socket

```
length_addr = sizeof(cli_addr);
getsockname(sockfd, (struct sockaddr*)&cli_addr, &length_addr);
            1                2                   3
```

1. Socket (criado anteriormente)

2. Endereço da estrutura que será preenchida
   Nota: ver explicação no ponto 5 da seção "Enviar mensagem"

3. Tamanho da estrutura definida no ponto 2

Configurar opções do socket

```
                1          2          3              4                5
setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(timeout));
```

1. Socket (criado anteriormente)

2. Nível em que a opção é definida.
   Nota: SOL_SOCKET *is the socket layer itself*

3. Opção que se que pretende definir
   SO_RCVTIMEO: define o timeout (em milissegundos) que a função recvfrom fica à espera.
   SO_BROADCAST: configura o socket para envios via Broadcast

4. Endereço do valor que se pretende definir

5. Tamanho do valor definido no ponto 4

Arrays de caracteres

- Em C não existe o conceito de string mas sim de array de caracteres que são terminados por um caracter nulo → \0

- É por este motivo que se tem **SEMPRE** que terminar a cadeia de caracteres recebidos pela função *recvfrom*

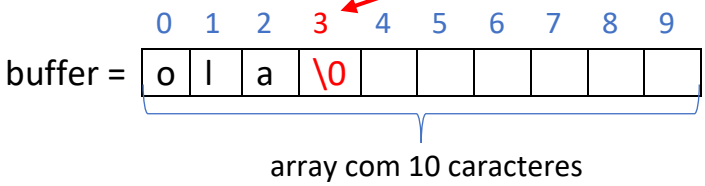
- Se não se terminar a cadeira de caracteres com o "\0", o resto do array será preenchido com lixo (aqueles 😊 que vimos nas aulas)

- Exemplo de um array de 10 caracteres quando se recebe a mensagem "ola":

```c
// Declarar um array de 10 caracteres
char buffer[10];

// Será recebida a mensagem "ola"
// o que significa que a variável nbytes terá o valor 3 (nº de bytes recebidos)
nbytes = recvfrom(sockfd, buffer, sizeof(buffer), 0, NULL, NULL);
if (nbytes == SOCKET_ERROR)
    Abort("Erro na recepcao de datagrams");

// Aceder à posição 3 (os arrays começam por zero) e colocar o '\0'
// de forma a terminar a cadeira de caracteres
buffer[nbytes]='\0';                    // buffer[3] = '\0';
printf("\n<SER1>Mensagem recebida {%s}\n", buffer);
```



array com 10 caracteres

Como o número de bytes recebidos é 3 (ola), então estamos a ir à posição 3 do buffer e colocar o "\0", de modo a terminar a cadeia de caracteres.