(IRC) Prática

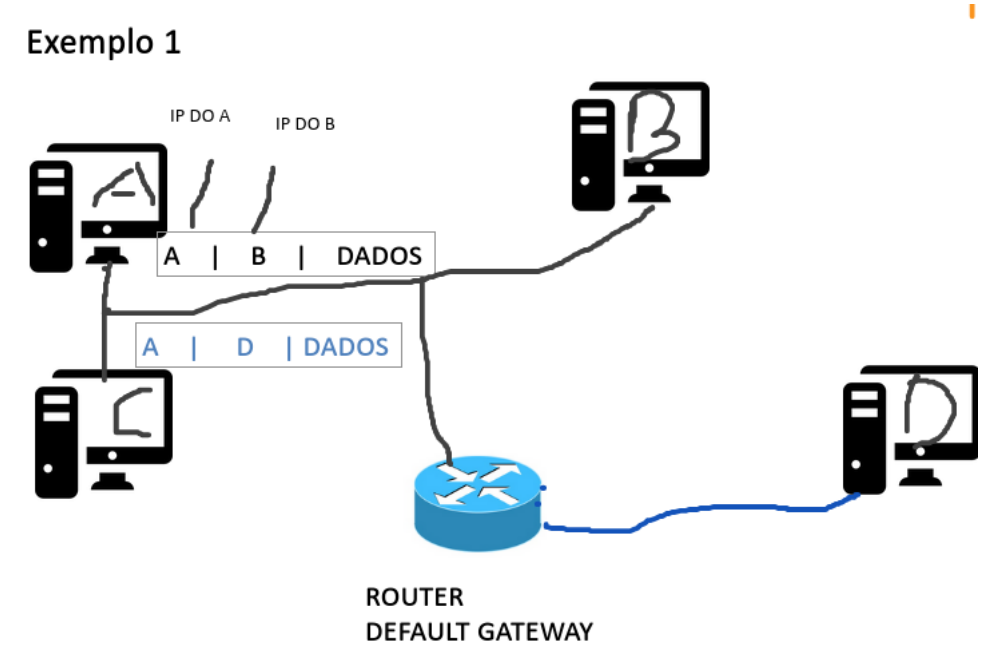
From WikiNote

Contents

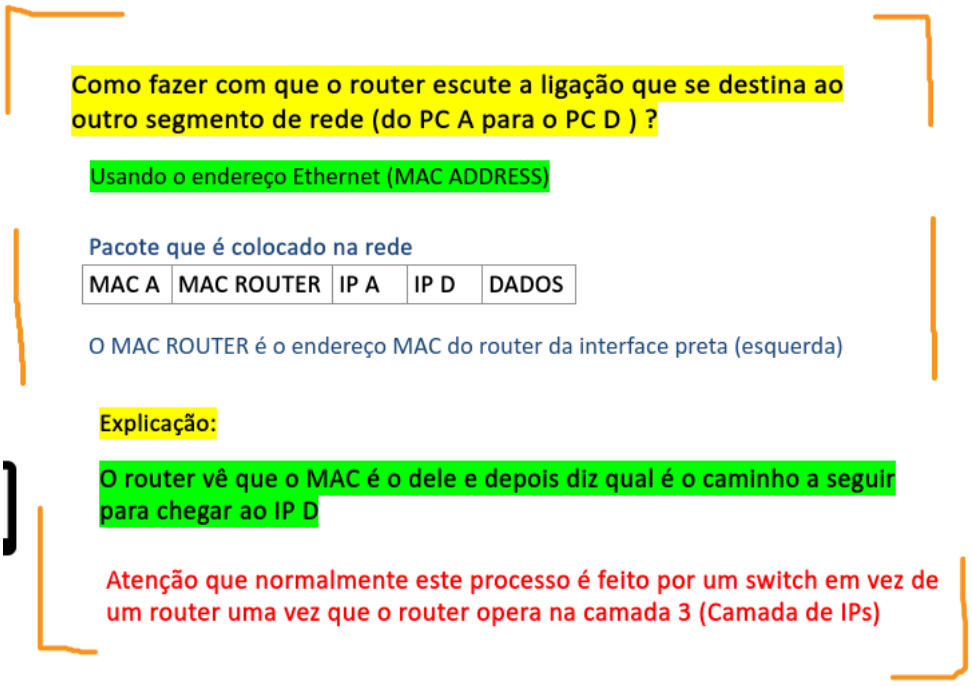
Aula Prática 13/10/2020

Exemplo 1

Esquema



Explicação



Depois do router saber que o pacote ethernet é para ele, ele remove o MAC A e o MAC Router (da interface esquerda) (porque os routers trabalham na camada 3 , IP), reconstroi o pacote e envia para o ip destino.

MAC ROUTER	MAC D	IP A	IP D	DADOS

O MAC Router é agora o MAC da interface azul do router (lado direito)

NOTA

O IP de origem tem de ir porque quando a outra máquina quiser responder, terá de saber quem enviou a mensagem.

Questões

Como é que a minha máquina sabe o MAC de Router?

Coloca um pacote na rede a perguntar , "quem tem este ip?" e o router responde "sou eu e toma o meu MAC ADDRESS".

No entanto para evitar estar sempre a perguntar, a nossa máquina cria uma tabela em memória para obter essa informação.

O comando é o **arp -a**.

Notas da Aula

As nossas aplicações

As nossas aplicações terão de se basear sempre num conjunto de 3 coordenadas pares para poderem funcionar.

- Protocolo
  - UDP
- Porto
  - Porto Origem
  - Porto Destino
- IP
  - IP Origem
  - IP Destino

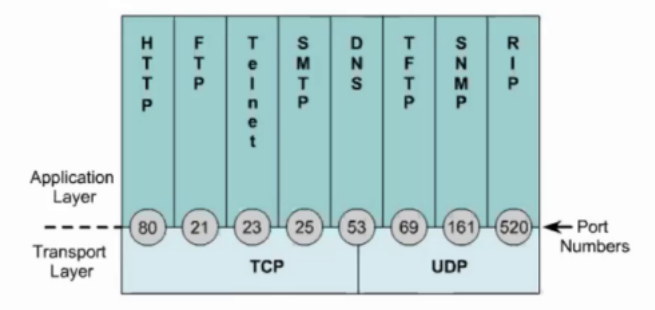
Os routers

Os routers trabalham sempre no nível 3 (Layer 3) (IP)

Well Known Ports

Todos os primeiros 1000 números são chamados Well Known Ports ou seja, já estão reservados para outros protocolos

TCP e UDP



O TCP funciona como um telefone

O UDP é como um sistema postal. Nunca há uma ligação permanente entre o recetor e o que envia.

- *User Datagram Protocol*
- Do tipo não orientado a ligação
- Não é fiável
- Preferível ao TCP quando não é exigida uma fiabilidade de 100%, devido à sua reduzida sobrecarga protocolar
- Exemplos
  - TFTP e DNS recorrem ao UDP e realizam os seus próprios controles de erros
  - Transmissão de voz ou de imagens digitalizadas em que erros ocasionais são aceitáveis

Aula Prática 27/10/2020

Criação do socket

Criação do socket

```
#include <sys/types.h>
#include <sys/socket.h>
socketfd = socket(protocol_family, socket_type, protocol);
```

int protocol_family

Identifica a família da pilha protocolar pretendida. Existem constantes simbólicas com os valores normalizados. Por exemplo: PF_INET, para o TCP/IP, PF_UNIX, para os protocolos internos do Unix, e PF_NS, para os serviços de rede Xerox.

int socket_type

Identifica o tipo de ligação pretendida:
SOCK_DGRAM para *datagrams* (canal não orientado a ligação); UDP
SOCK_STREAM para circuito virtual (canal orientado a ligação); TCP
SOCK_RAW para os protocolos de baixo nível normalmente utilizados pela rede (e.g., ICMP).

int protocol

Identifica o protocolo que pretendemos usar. Devemos fornecer uma das constantes simbólicas presentes no ficheiro <netinet/in.h>. Para a família protocolar TCP/IP alguns dos valores possíveis são IPPROTO_TCP, IPPROTO_UDP, IPPROTO_IP, IPPROTO_ICMP. Este valor normalmente não é especificado, sendo usado o valor 0 (zero) que obriga o próprio sistema operativo a seleccionar o protocolo adequado. Repare-se que, no caso do TCP/IP, se for conhecida a família protocolar e o tipo de canal que estamos a requisitar é possível inferir qual o protocolo de transporte a usar.

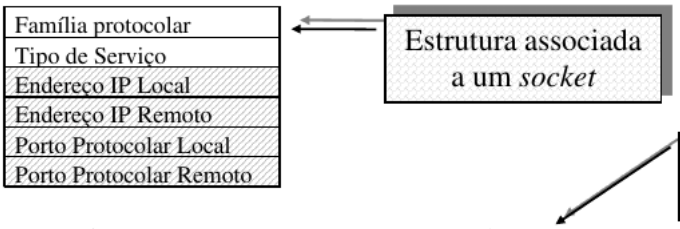
int socketfd RETURN

-1 se ocorrer erro ou um valor diferente representando o descritor para o *socket* criado.

Exemplo

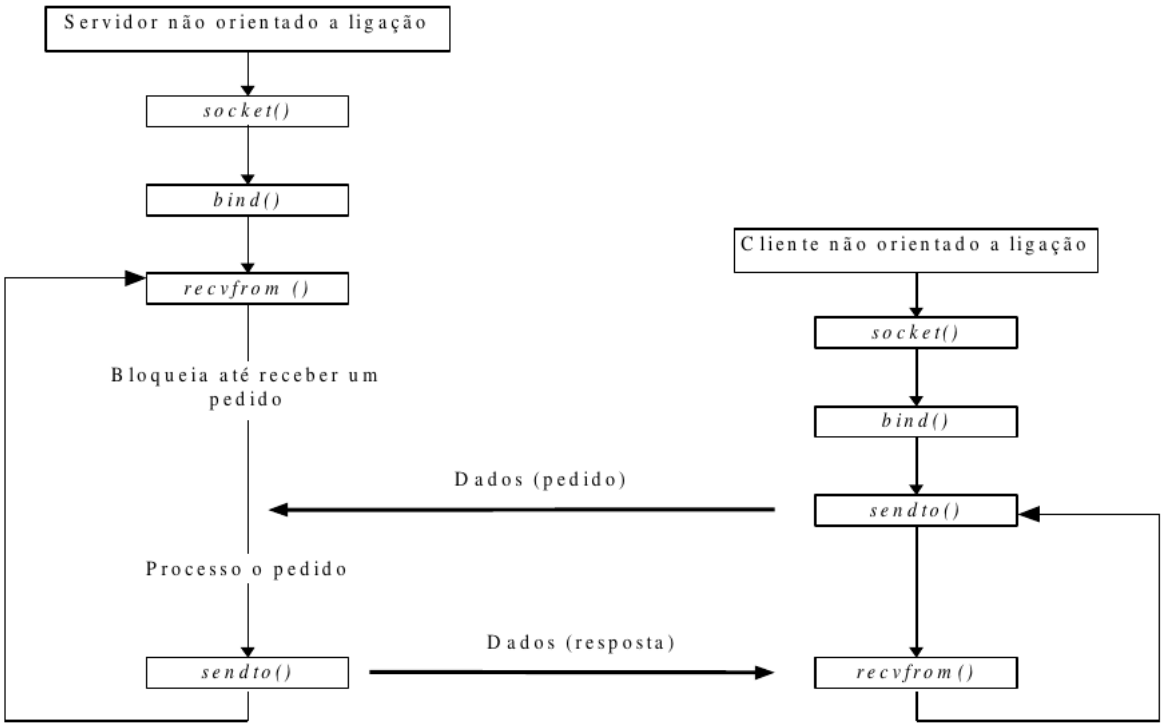
```
Socketfd = socket(AF_INET, SOCK_DGRAM, 0) // Cria socket UDP
Socketfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); // Cria socket TCP
```

Configuração do socket



Objectivo do *Socket*	Informação Local	Informação Remota
Cliente orientado a ligação	Uma chamada à rotina *connect()* tem por efeito armazenar toda a informação local e remota na estrutura de dados correspondente.	
Servidor orientado a ligação	*bind()*	*listen* e *accept()*
Cliente não orientado a ligação	*bind()*	*sendto()*
Servidor não orientado a ligação	*bind()*	*recvfrom()*

UDP



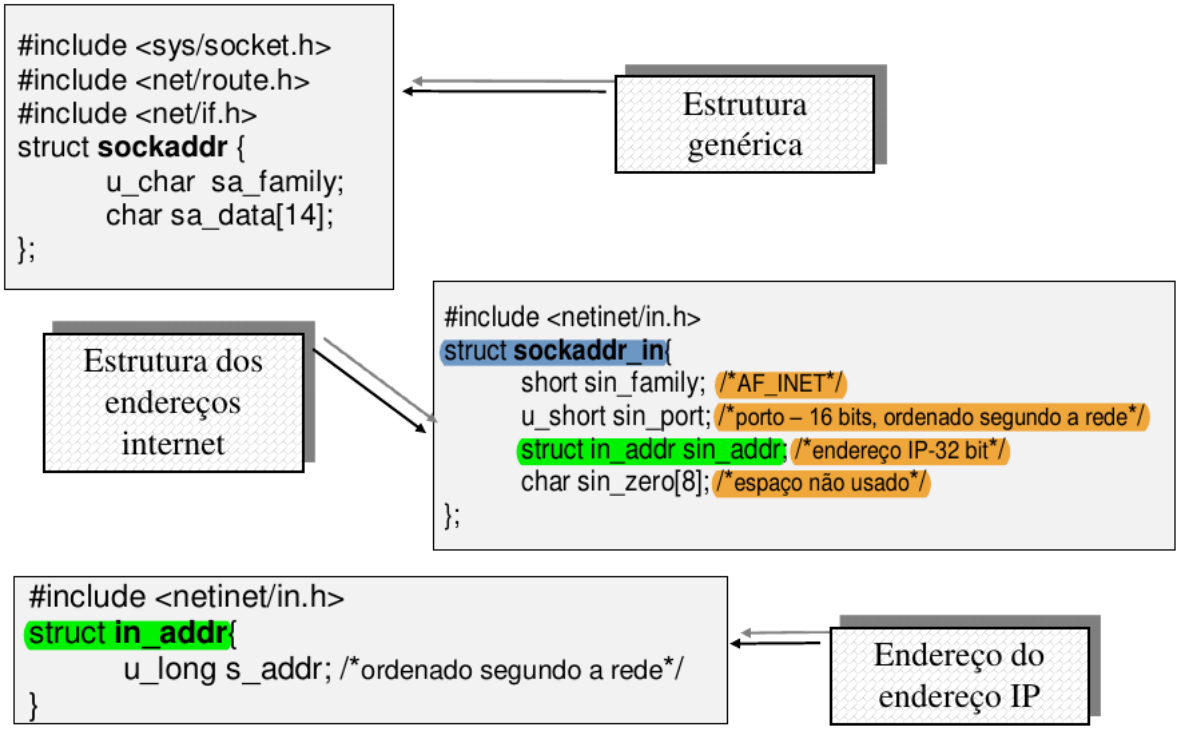
bind()

bind()

```
#include <sys/types.h>
#include <sys/socket.h>
result = bind (socketfd, addr, addr_len);
```

int socketfd
Trata-se do descritor fornecido pelo SO através da chamada *socket()*.
struct sockaddr * addr
Ponteiro para uma estrutura onde é armazenado o endereço que se pretende vincular à nossa aplicação.
int addr_len
Tamanho, em *bytes*, da estrutura apontada por **addr**.
int result
-1 se ocorrer erro ou 0 caso contrário.

Estrutura do endereçamento



sendto()

#include <sys/types.h>
#include <sys/socket.h>
result = **sendto** (socketfd, msg, len, flags, to, tolen)

int socketfd
Trata-se do descritor fornecido pelo SO através da chamada *socket()* sobre o qual pretendemos transmitir dados.
const void *msg
Ponteiro para a zona de memória onde se encontram os *bytes* a transmitir.
size_t len
Número de *bytes* a transmitir.
int flags
Servem para configurar algumas questões de baixo nível no modo de envio dos dados.
const struct sockaddr *to
Endereço do destinatário dos dados.
int tolen
Tamanho do endereço do destinatário dos dados.
int result
-1 se ocorrer algum erro ou o número de *bytes* enviados caso contrário.

recvfrom()

recvfrom()

#include <sys/types.h>
#include <sys/socket.h>
result = **recvfrom** (socketfd, msg, len, flags, from, fromlen)

int socketfd
Trata-se do descritor, fornecido pelo SO através da chamada *socket()*, sobre o qual pretendemos receber dados.
const void *msg
Ponteiro para a zona de memória onde se pretende guardar os *bytes* recebidos.
size_t len
Número de *bytes* máximo a receber.
int flags
Servem para configurar algumas questões de baixo nível na recepção dos dados.
const struct sockaddr * from
Se este ponteiro não for *null* então é copiado para o endereço por ele apontado o endereço do emissor (apenas se o *socket* disser respeito a uma comunicação não orientada a ligações).
int fromlen
Tamanho do endereço apontado por "from".
int result
-1 se ocorrer algum erro ou o número de *bytes* recebidos caso contrário.

sendmsg()

sendmsg()

#include <sys/types.h>
#include <sys/socket.h>
result=**sendmsg** (socketfd, msg, flags)

int socketfd
Descritor, fornecido pelo SO através da chamada *socket()*, sobre o qual pretendemos transmitir dados.
const struct msghdr *msg
Ponteiro para a estrutura onde se armazenam os elementos necessários à transmissão.
int flags
Servem para configurar algumas questões de baixo nível no modo de envio dos dados.
int result
-1 se ocorrer algum erro ou o número de *bytes* enviados caso contrário.

Ficha 1, exercicio 1

clienteUDP_v1

```c
#define SERV_HOST_ADDR "127.0.0.1"
#define SERV_UDP_PORT  6000

#define BUFFERSIZE     4096

void Abort(char *msg);

/*_____________________________ main ____________________________________
*/

int main( int argc , char *argv[] )
{

    SOCKET sockfd;                  // criamos o socket
    int msg_len, iResult;           //msg_len permite obter o tamanho da mensagem a enviar
    struct sockaddr_in serv_addr;   // serv_addr é onde vamos colocar o endereço e porto do
    char buffer[BUFFERSIZE];
    WSADATA wsaData;

    /*========================= TESTA A SINTAXE =========================*/

    if(argc != 2){
        fprintf(stderr,"Sintaxe: %s frase_a_enviar\n",argv[0]);
        exit(EXIT_FAILURE);
    }
    /*=============== INICIA OS WINSOCKS ==============*/

    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup failed: %d\n", iResult);
        getchar();
        exit(1);
    }

    /*=============== CRIA SOCKET PARA ENVIO/RECEPCAO DE DATAGRAMAS ==============*/

    sockfd = socket( PF_INET , SOCK_DGRAM , 0 );
    if(sockfd == INVALID_SOCKET)
        Abort("Impossibilidade de criar socket");

    /*================= PREENCHE ENDERECO DO SERVIDOR ====================*/

    memset( (char*)&serv_addr , 0, sizeof(serv_addr) ); /*Coloca a zero todos os bytes (limp
    serv_addr.sin_family = AF_INET; /*Address Family: Internet*/
    serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR); /*IP no formato "dotted decimal"
                                                            inet_addr converte para o valor b
    serv_addr.sin_port = htons(SERV_UDP_PORT); /*Host TO Network Short*/

    /*====================== ENVIA MENSAGEM AO SERVIDOR ==================*/

    msg_len = strlen(argv[1]); // temos de saber qual é o tamanho da mensagem

    if(sendto( sockfd , argv[1] , msg_len , 0 , (struct sockaddr*)&serv_addr , sizeof(serv_a
        Abort("SO nao conseguiu aceitar o datagram");

    printf("<CLI1>Mensagem enviada ... a entrega nao e' confirmada.\n");

    /*========================= FECHA O SOCKET ===========================*/

    closesocket(sockfd);

    printf("\n");
    exit(EXIT_SUCCESS);
```

```
}

/*_____________________________ Abort_____________________________________
  Mostra uma mensagem de erro e o código associado ao ultimo erro com Winsocks.
  Termina a aplicacao com "exit status" a 1 (constante EXIT_FAILURE)
_____________________________________________________________________________*/

void Abort(char *msg)
{

    fprintf(stderr,"<CLI1>Erro fatal: <%s> (%d)\n",msg, WSAGetLastError());
    exit(EXIT_FAILURE);

}
```

servidorUDP_v1

```c
#define SERV_UDP_PORT 6000
#define BUFFERSIZE     4096

void Abort(char *msg);

/*_____________________________ main ____________________________________
*/

int main( int argc , char *argv[] )
{
    SOCKET sockfd;                  //criamos um socket
    int iResult, nbytes;            //nbytes diz o nr de bytes recebidos
    struct sockaddr_in serv_addr;   //serv_addr é a estrutura que vai ficar com a informaçã
    char buffer[BUFFERSIZE];
    WSADATA wsaData;

    /*=============== INICIA OS WINSOCKS ==============*/

    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup failed: %d\n", iResult);
        getchar();
        exit(1);
    }
    /*============ CRIA O SOCKET PARA RECEPCAO/ENVIO DE DATAGRAMAS UDP ============*/

    if((sockfd = socket( PF_INET , SOCK_DGRAM , 0)) == INVALID_SOCKET)
        Abort("Impossibilidade de abrir socket");

    /*=============== ASSOCIA O SOCKET AO  ENDERECO DE ESCUTA ===============*/

    /*Define que pretende receber datagramas vindos de qualquer interface de
    rede, no porto pretendido (6000)*/

    memset( (char*)&serv_addr , 0, sizeof(serv_addr) );  // limpamos a estrutura associada a
    serv_addr.sin_family = AF_INET; /*Address Family: Internet*/
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);  /*Host TO Network Long
                                                      INADDR_ANY -> recebemos informação vinda
    serv_addr.sin_port = htons(SERV_UDP_PORT);  /*Host TO Network Short*/

    /*Associa o socket ao porto pretendido (bind())*/
    if(bind( sockfd , (struct sockaddr *)&serv_addr , sizeof(serv_addr)) == SOCKET_ERROR)
        Abort("Impossibilidade de registar-se para escuta");

    /*================ PASSA A ATENDER CLIENTES INTERACTIVAMENTE =============*/

    while(1){

        fprintf(stderr,"<SER1>Esperando datagram...\n");

        nbytes=recvfrom(sockfd , buffer , sizeof(buffer) , 0 , NULL , NULL);
        /*o buffer serve para guardar a informação recebida*/

        /*nbytes será o nr de bytes recebidos*/
        if(nbytes == SOCKET_ERROR)
            Abort("Erro na recepcao de datagrams");

        buffer[nbytes]='\0'; /*Termina a cadeia de caracteres recebidos com '\0'
                               porque as mensagens recebidas não estão terminadas*/

        printf("\n<SER1>Mensagem recebida {%s}\n",buffer);
```

```
    }

}
```

Ficha 1, exercicio 2

clienteUDP_v2

No cliente, a única coisa que alteramos foi a receção de mensagens do servidor. Ou seja, enviamos a mensagem ao servidor e depois recebemos a mensagem vinda do servidor de confirmação que foi enviada.

```
/*========================= RECEBE MENSAGEM DO SERVIDOR ============================*/


int nbytes;

/*Vamos receber agora a resposta do servidor com o recvfrom()*/
nbytes = recvfrom(sockfd, buffer, sizeof(buffer), 0, NULL, NULL);


if (nbytes == SOCKET_ERROR) {
    Abort("Erro na receção do datagrama");
}

/*Se nao der erro, terminamos a string com \0 uma vez que este processo não é automatico*/
buffer[nbytes] = '\0';

printf("<CLI1> Mensagem recebida : {%s}\n", buffer);
```

servidorUDP_v2

No servidor temos de receber a mensagem e enviar para o cliente a confirmação do envio dessa mensagem.

No recvfrom , alteramos alguns parâmetros uma vez que precisamos de guardar a informação sobre o cliente e saber qual é o tamanho da estrutura do cliente (este tamanho tem de ser passado por referência).

O que foi alterado está a amarelo sublinhado.

```
/*================ PASSA A ATENDER CLIENTES INTERACTIVAMENTE =============*/
struct sockaddr_in cli_addr;     // precisamos de uma estrutura sockaddr_in para guardar o er
int cli_len;                     // precisamos de um int para guardar o tamanho da estrutura

cli_len = sizeof(cli_addr);      // colocamos o tamanho da estrutura cli_addr no cli_len

while(1){

    fprintf(stderr,"<SER1>Esperando datagrama...\n");


    /*No recvfrom() temos de guardar o endereço do cliente para enviar a resposta de confirma
    nbytes=recvfrom(sockfd , buffer , sizeof(buffer) , 0 ,(struct sockaddr *) &cli_addr, &cli

    if(nbytes == SOCKET_ERROR)
        Abort("Erro na recepcao de datagrams");

    buffer[nbytes]='\0'; /*Termina a cadeia de caracteres recebidos com '\0'*/

    printf("\n<SER1>Mensagem recebida {%s}\n",buffer);

    /*Agora temos de enviar a mensagem para o cliente a confirmar*/
    if (sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *) &cli_addr, cli_len) ==
        Abort("O SO não conseguiu aceitar o datagrama");
    }

    printf("<SER1> Mensagem Confirmada \n");
}
```

Aula Prática 03/11/2020

Ficha 1, exercicio 3

Ha sempre um porto no lado do cliente e um porto no lado do servidor Como o cliente não fez nenhum bind() só lhe é atribuido um porto depois do mesmo fazer um sendto()

Como é que vamos obter a informação do porto do cliente?

Depois do cliente enviar a mensagem, temos de chamar a função getsockname

*** int getsockname(sockfd, addr, addr_len)**

*** int sockfd -> socket**

*** struct sock_addr *addr -> ponteiro para a estrutura onde fica a informação da nossa aplicação**

*** int addr_len -> tamanho em bytes da estrutura sock_addr *addr**

*** Esta função devolve, SOCKET_ERROR se ocorrer um erro, 0 caso contrário.**

- **No entanto temos de converter o porto**
  - **htons -> host to network short , queremos o contrario, ou seja, network short to host (ntohs)**

```
/* Acrescentamos isto ao cliente*/

int info_len = sizeof(info_addr);
if (getsockname(sockfd, (struct sockaddr*)&info_addr, &info_len) == SOCKET_ERROR)
    Abort("Erro na obtenção do porto\n");

printf("Porto : %d", ntohs(info_addr.sin_port));
/* Até aqui*/
```

Quando é para enviar informação dos portos por exemplo, usa-se o porto no formato host to network short (htons) e quando querem receber é sempre ao contrario, ou seja, network to host short (ntohs)

Ficha 1, exercicio 4

Quando fazemos o rcvfrom no servidor, já temos a estrutura do cliente que é a informação que queremos para fazermos o exercicio

- cli_addr.sin_port -> porto do cliente em formato host to network short
- cli_addr.sin_addr -> ip do cliente mas em formato binário (temos de converter de binário para dot.decimal)
  - inet_addr -> converte de "127.0.0.1" para binário

Como temos de alterar o ip do cliente de binário para doted decimal, usamos a função inet_ntoa().

```
printf("Ip do cliente: %s, Porto do cliente: %d\n",inet_ntoa(cli_addr.sin_addr),ntohs(cli_addr.sin_port));
```

Aula Prática 10/11/2020

Ficha1, Exercicio 5

```
Costumavamos chamar assim : cliente.exe "Ola Mundo"
Agora queremos chamar assim : cliente.exe -msg "Ola Mundo" -ip 127.0.0.1 -port 6000
OU                cliente.exe -ip 127.0.0.1 -port 6000 -msg "Ola Mundo"
OU                cliente.exe -port 6000 -ip 127.0.0.1 -msg "Ola Mundo"
```

O nr de argumentos do cliente passa de 2 para 7

```
/*========================= TESTA A SINTAXE =========================*/

if (argc != 7) {
    fprintf(stderr, "Sintaxe: %s -msg frase_a_enviar -ip ip_destino -port porto_destino\n"
    exit(EXIT_FAILURE);
}
```

```
/*=============== TRATA OS ARGUMENTOS ==============*/
int ip_idx, port_idx, msg_idx;
for (int i = 0; i < 7; i++) {
    if (strcmp(argv[i],"-ip") == 0) {
        ip_idx = i + 1;
    }
    if (strcmp(argv[i], "-port") == 0) {
        port_idx = i + 1;
    }
    if (strcmp(argv[i], "-msg") == 0) {
        msg_idx = i + 1;
    }

}
```

Desta maneira conseguimos aceitar qualquer order de argumentos, desde que o cliente passe os argumentos corretos

```
  /*================= PREENCHE ENDERECO DO SERVIDOR ====================*/

  memset((char*)&serv_addr, 0, sizeof(serv_addr)); /*Coloca a zero todos os bytes*/
  serv_addr.sin_family = AF_INET; /*Address Family: Internet*/
  serv_addr.sin_addr.s_addr = inet_addr(argv[ip_idx]); /*IP no formato "dotted decimal" => 32
  serv_addr.sin_port = htons(atoi(argv[port_idx])); /*Host TO Netowork Short*/

/*===================== ENVIA MENSAGEM AO SERVIDOR ==================*/

  msg_len = strlen(argv[msg_idx]);

  if (sendto(sockfd, argv[msg_idx], msg_len, 0, (struct sockaddr*)&serv_addr, sizeof(serv_add
      Abort("SO nao conseguiu aceitar o datagram");
```

Temos de converter de string para inteiro

Descobrimos o tamanho do conteudo da -msg

Enviamos o conteudo que esta na flag -msg

Ficha1, Exercicio 6

No rcvfrom do cliente devemos colocar uma estrutura para guardar a informação de quem nos enviou a mensagem e depois temos de comparar se o Ip e Porto que essa estrutura guardou é ou não igual ao Ip e Porto do servidor (que está guardado na estrutura serv_addr)

Para sabermos se não é o impostor:

- O IP de origem tem de ser igual ao ip do servidor e o Porto de origem tem de ser igual a 6000(porto do servidor)

Conseguimos comparar os portos porque o valor que lá está escrito é o mesmo (binário) No entanto não conseguimos comparar os endereços, temos de converter os endereços para dot decimal e fazer um strcmp (usamos o inet_ntoa e o strcmp)

```
struct sockaddr_in check;
int check_len = sizeof(check);

int nbytes;
nbytes = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&check,&check_len);


if (check.sin_port == serv_addr.sin_port && strcmp(inet_ntoa(check.sin_addr), inet_ntoa(serv_
    printf("\nMensagem veio do servidor\n");
}
else {
    printf("\nMensagem veio de um impostor\n");
}
```

NOTA IMPORTANTE

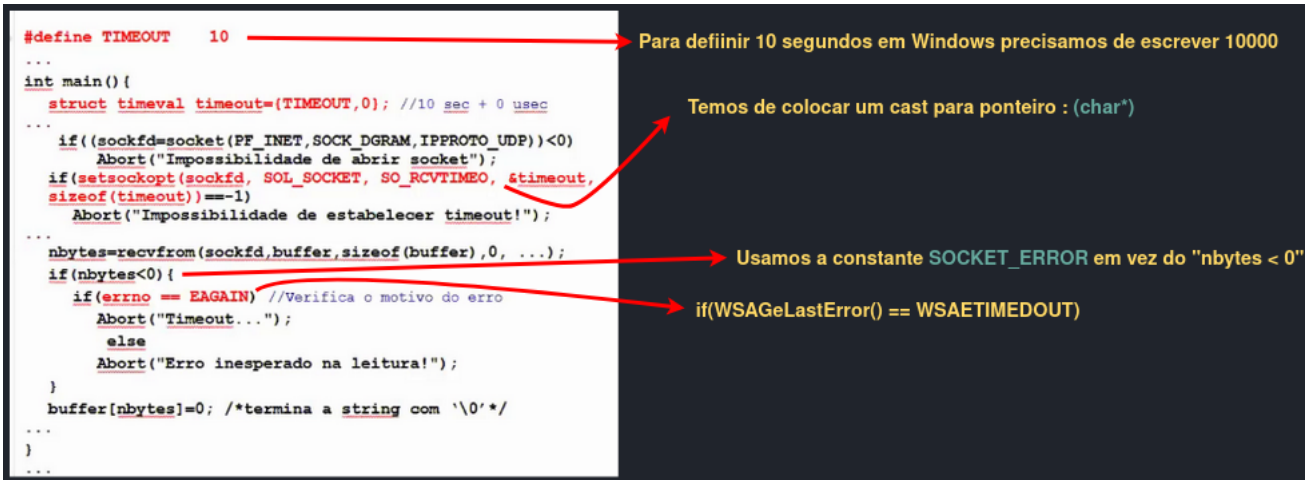
Quando é para enviar informação dos portos por exemplo, usa-se o porto no formato host to network (htons) e quando queremos receber é sempre ao contrario ou seja, network to host (ntohs).

Aula Prática 17/11/2020

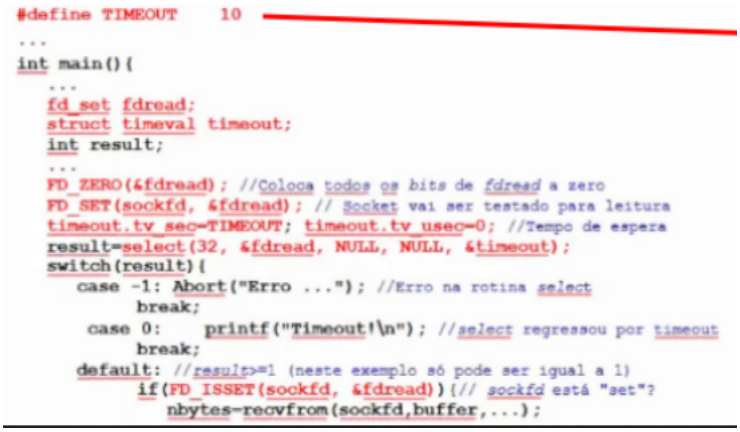
Timeout de Receção

- Limitar o tempo de espera pela recepção de um datagrama (UDP) ou de segmentos (TCP)
- Várias soluções
  - sinal ***SIGALRM*** e rotinas ***signal()*** e ***alarm()***
  - rotina ***setsockopt()***
  - rotina ***select()***
    - Multiplexagem de Entrada/Saída

Implementação do setsockopt



Usando o select



Ficha1, Exercicio 8

Usando o setsockopt

```
/*================ FAZ SET DO TIMEOUT ==============*/
/*================ ATRIBUI UM TIMEOUT AO SOCKET ==============*/

struct timeval timeout = { TIMEOUT, 0 };

if (setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(timeout)) == -1)
    Abort("Impossibilidade de estabelecer timeout!");
```

```
/*=============== VERIFICO SE HOUVE ERRO NO TIMEOUT ==============*/


if (nbytes == SOCKET_ERROR) {
    if (WSAGetLastError() == WSAETIMEDOUT)
        Abort("Timeout\n");
    else
        Abort("Erro inesperado na leitura\n");

    Abort("Erro na recepcao de datagrams");
}
```

Usando o select

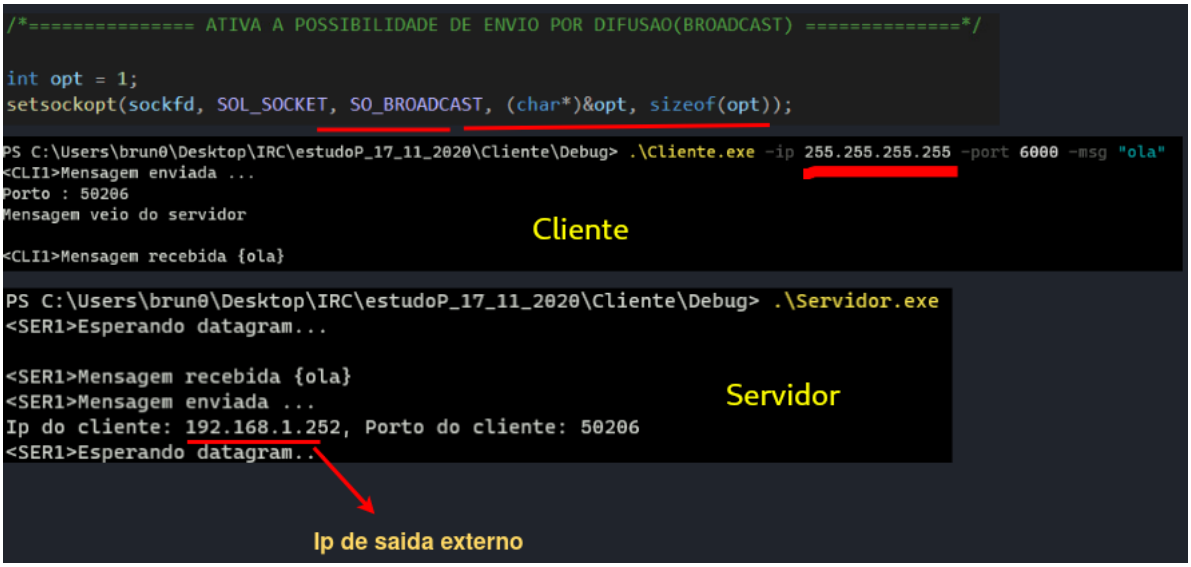
```
/*=============== CRIA TIMEOUT USANDO O SELECT==============*/
struct timeval timeout = { TIMEOUT,0 };
fd_set fd_read;
int result;

FD_ZERO(&fd_read);
FD_SET(sockfd, &fd_read);
```

```
result = select(32, &fd_read, NULL, NULL, &timeout);

switch (result) {
case -1:Abort("Erro ..."); // Erro na rotina select
    break;
case 0: printf("Timeout!\n"); // select regressou por timeout
    break;
default: // result == 1
    if (FD_ISSET(sockfd, &fd_read)) {
        nbytes = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&check, &check_len);

        if (nbytes == SOCKET_ERROR)
            Abort("Erro na recepcao de datagrams");

        buffer[nbytes] = '\0'; /*Termina a cadeia de caracteres recebidos com '\0'*/

        printf("\n<CLI1>Mensagem recebida {%s}\n", buffer);

        if (check.sin_port == serv_addr.sin_port && strcmp(inet_ntoa(check.sin_addr), inet_ntoa(serv_addr.sin_ad
            printf("\nMensagem veio do servidor\n");
        }
        else {
            printf("\nMensagem veio de um impostor\n");
        }
    }
}
```

Ficha1, Exercicio 7 (adaptado)

Caso eu quisesse utilizar o ip de broadcast (255.255.255.255) no Cliente iria ter um erro de Permission enied representado pelo WSAEACCES (10013).

Para que isto seja possivel, tenho de fazer o que mostra o código abaixo



Ficha1, Exercicio 9

Neste caso só precisamos de alterar o **sendto()** do servidor uma vez que o cliente já envia o tamanho da sua mensagem.

Já nao vamos usar o **buffer** mas vamos criar um **char resposta[5]**

Depois fazemos **sprintf(resposta,"%d", strlen(buffer));** e enviamos a resposta no **sendto()**

Ficha1, Exercicio 10

No servidor criamos um **int resposta** e no **sendto()** alteramos algumas coisas ...

resposta = strlen(buffer);

sendto(sockfd,(char *)&resposta, sizeof(resposta) ...);

No Cliente temos de alterar o **recvfrom()** ...

int resposta

recvfrom(sockfd,(char*)&reposta,sizeof(resposta),..)

Ficha 1, Exercicio 11

No servidor

```
int cli_1_len = sizeof(cli_1_addr);
int cli_2_len = sizeof(cli_2_addr);
while (1) {

    fprintf(stderr, "<SER1>Esperando datagrama...\n");

    nbytes = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&cli_1_addr, &cli_1_len);

    if (nbytes == SOCKET_ERROR)
        Abort("Erro na recepcao de datagrams");

    nbytes = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)&cli_2_addr, &cli_2_len);

    if (nbytes == SOCKET_ERROR)
        Abort("Erro na recepcao de datagrams");

    printf("\n<SER1>Mensagem recebida {%s}\n", buffer);


    /*================ ENVIA MENSAGEM DO CLIENTE 2 PARA O CLIENTE 1=============*/



    if (sendto(sockfd, (char *)&cli_1_addr, sizeof(cli_1_addr), 0, (struct sockaddr*)&cli_2_addr, cli_2_len) == SOCKET_ERROR) {
        Abort("[ERRO]");
    }

    printf("<SER1> Par formado\n");
}
```

No cliente

```c
/*Estamos à espera de receber uma estrutura do tipo sockaddr_in*/
nbytes = recvfrom(sockfd, (char *)&cli_addr, cli_len, 0, (struct sockaddr*)&info_addr, &info_addr_len);

if (nbytes == SOCKET_ERROR)
    Abort("[ERRO]");

if (nbytes != cli_len) {
    Abort("Erro inesperado");
}

if (strcmp(argv[ip_des], inet_ntoa(info_addr.sin_addr)) == 0 && atoi(argv[port_des]) == ntohs(info_addr.sin_port)) // mensagem veio do servidor
{
    // o cliente envia mensagem para o seu par
    nbytes = sendto(sockfd, (char*)&cli_addr, cli_len, 0, (struct sockaddr*)&cli_addr, cli_len);

    if (nbytes == SOCKET_ERROR)
        Abort("[ERRO]");

    printf("Sou o cliente 2\n");
    printf("O meu par remoto tem o IP %s e o Porto %d", inet_ntoa(cli_addr.sin_addr), ntohs(cli_addr.sin_port));
}
// mensagem veio do cliente 1
else {
    printf("Sou o cliente 1\n");
    printf("Par remoto tem o IP %s e o Porto %d", inet_ntoa(info_addr.sin_addr),ntohs(info_addr.sin_port));
}


/*======================== FECHA O SOCKET ==========================*/

closesocket(sockfd);

exit(EXIT_SUCCESS);
```

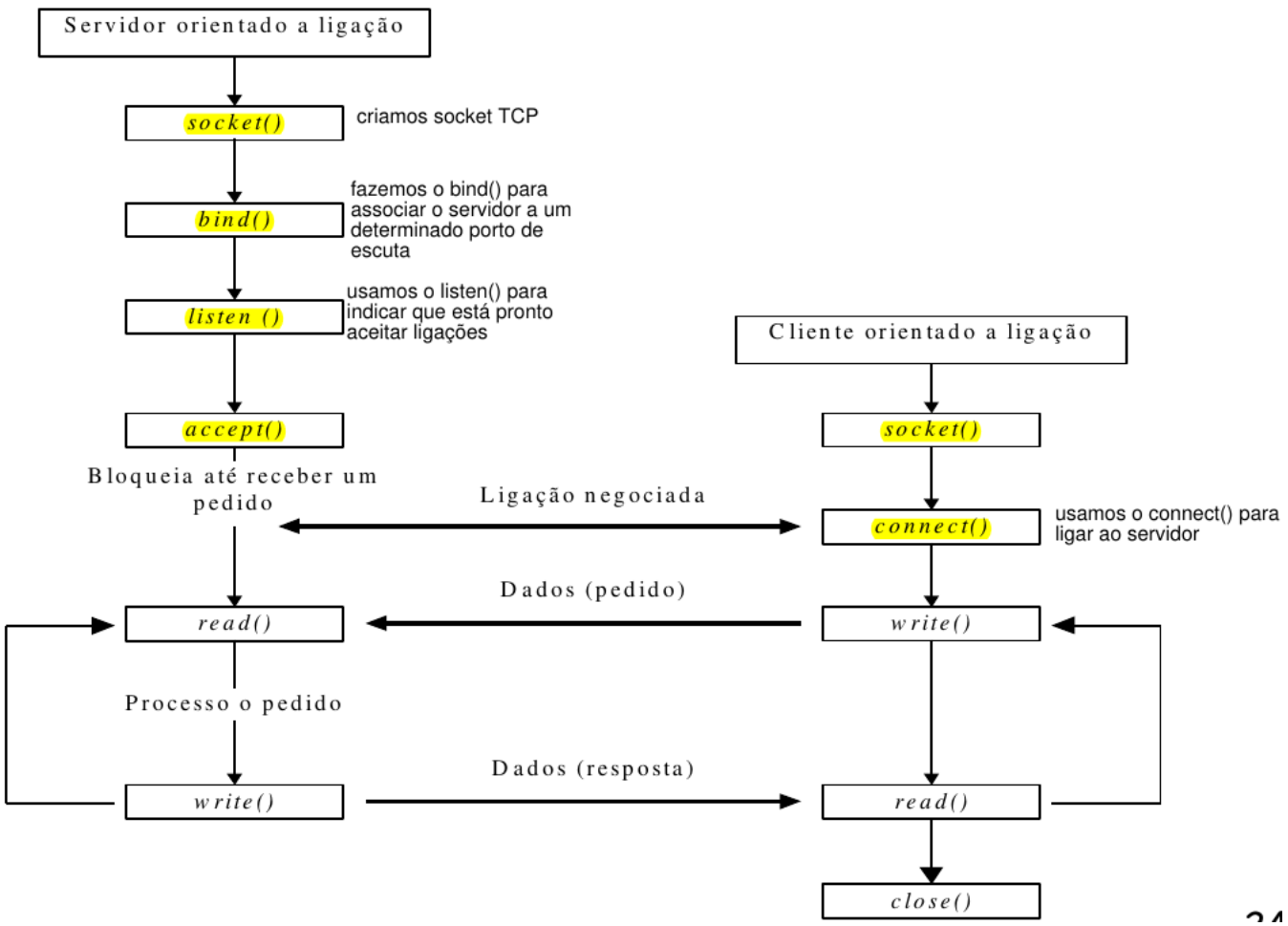
Guardamos quem enviou a mensagem

Nesta estrutura estará o cliente 1

Aula 22-12-2020

O TCP é orientado à ligação e à confirmação da mensagem

Fluxo em sistemas TCP



listen()

listen()

```
#include <sys/socket.h>
result=listen(socketfd, backlog);
```

int socketfd

Trata-se do descritor, fornecido pelo SO através da chamada *socket()*, sobre o qual pretendemos receber pedidos de ligação.

int backlog

Representa o número máximo de ligações pendentes (em fila de espera) que o SO deve aceitar. O valor máximo de ligações que os SO normalmente impõe é 5.

int result

-1 se ocorrer erro ou 0 caso contrário.

```
struct sockaddr_in serv_addr;

/*================== PREENCHE ENDERECO DE ESCUTA ======================*/
bzero((char*)&serv_addr,sizeof(serv_addr));
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);  /*Recebe de qq interface*/
serv_addr.sin_port=htons(SERV_TCP_PORT);  /*Escuta no porto definido*/

/*====================== REGISTA-SE PARA ESCUTA ========================*/
if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) <0)
        Abort("Impossibilidade de registar-se para escuta");

if(listen(sockfd,5)==-1) Abort("Impossibilidade de escutar pedidos");
```

accept()

accept()

```
#include <sys/types.h>
#include <sys/socket.h>
result=accept(socketfd, addr, addrlen);
```

int socketfd

Descritor, fornecido pelo SO através da chamada *socket()*, sobre o qual pretendemos receber pedidos de ligação.

struct sockaddr *addr

É um ponteiro para o endereço onde será armazenado o endereço do cliente.

int *addrlen

Valor/resultado onde se passa o tamanho, em *bytes*, do endereço apontado por "addrlen".

int result

-1 se ocorrer erro ou valor positivo representando um descritor para o novo *socket* representado o extremo servidor da ligação estabelecida.

connect()

connect()

```
#include <sys/types.h>
#include <sys/socket.h>
result=connect(socketfd, addr, addrlen)
```

int socketfd

Descritor, fornecido pelo SO através da chamada *socket()*, sobre o qual pretendemos estabelecer uma ligação.

struct sockaddr *addr

É um ponteiro para o endereço onde está armazenado o endereço do servidor.

int *addrlen

Valor/resultado onde se passa o tamanho, em *bytes*, do endereço apontado por "addrlen".

int result

-1 se ocorrer erro ou 0 caso contrário.

```
if(connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))==-1)
                Abort("Impossibilidade de estabelecer ligacao");
```

close() & shutdown()

close() & shutdown()

#include <unistd.h>
result = **close** (sockfd);

int sockfd
Descritor do *socket* da sessão que pretendemos terminar.
int result
-1 se o ocorrer algum erro, 0 em caso contrário.

#include <sys/socket.h>
result = **shutdown** (int sockfd, int how);

int sockfd
Descritor do *socket* da sessão que pretendemos terminar.
int how
Modo como desejamos terminar a sessão *full-duplex*: 0- cancelamento de recepção 1- cancelamento de envios 2- cancelamento de recepção e envios
int result
-1 se o ocorrer algum erro, 0 em caso contrário.

Tipos de servidores TCP

- Existem dois tipos de servidor:
  - **O servidor interativo** que atende um cliente de cada vez
    - Se o **acept()** devolver um valor válido, atendemos o cliente e no final desse atendimento fechamos o socket desse cliente e só depois é que podemos aceitar outra ligação
  - **O servidor concorrente** lança um processo filho para atender cada cliente recebido

Troca de Dados

write()	Envia *array* de *bytes*.
read ()	Recebe *array* de *bytes*.
send()	Envia *array* de *bytes* podendo definir modo de envio.
receive()	Recebe *array* de *bytes* podendo definir modo de envio.
writev()	Envia mensagem estruturada.
readv()	Recebe mensagem estruturada.

read() & write()

#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
result = **read** (socketfd, buf, nbytes);
result = **write** (socketfd, buf, nbytes);

int socketfd
Descritor do *socket* sobre o qual pretendemos receber/enviar dados.
void *buf
Ponteiro para a zona de memória de dados a receber/enviar.
int nbytes
Número de *bytes* máximo a receber/enviar.
int result
-1 se ocorrer algum erro, 0 se EOF (apenas no *read()*), ou o número de *bytes* recebidos/enviados.

Exercicio 2 ,3 e 5 Ficha TCP

na pasta aulaP_22_12_2020

DNS

gethostbyname() & gethostbyaddr()

gethostbyname() & gethostbyaddr()

Nome ⇒ *IP*

#include <netdb.h>
result = **gethostbyname** (name);

const char * name
Nome da máquina para a qual se quer operar a conversão.
struct hostent * result
NULL se o ocorrer algum erro, um ponteiro para uma estrutura "*hostent*" caso contrário.

IP ⇒ Nome

#include <netdb.h>
result = **gethostbyaddr** (addr, len, type);

const char * addr
Ponteiro para uma estrutura "addr_in".
int len
Tamanho do endereço.
int type
Tipo do endereço (AF_INET).
struct hostent *result
NULL se o ocorrer algum erro, um ponteiro para uma estrutura "*hostent*" caso contrário.

hostent & h_errno

```
#include <netdb.h>
struct hostent {
        char * h_name;       /* nome oficial (canónico) da máquina */
        char ** h_aliases;   /* lista de nomes alternativos */
        int h_addrtype;      /* tipo de endereço (AF_INET) */
        int h_length;        /* tamanho do endereço (4 bytes) */
        char ** h_addr_list; /* lista de endereços devolvidos */
}
```

```
#include <netdb.h>
extern int h_errno;
  // HOST_NOT_FOUND
  // TRY_AGAIN
  // NO_RECOVERY
  // NO_DATA

herror (string);
```

const char * string
Afixa a string (se não for *Null*) seguida da mensagem de erro ocorrida.

Fontes

- https://www.installsetupconfig.com/win32programming/windowsocketwinsock214index.html
- https://www.winsocketdotnetworkprogramming.com/winsock2programming/winsock2advancedcode1a.html
- https://stackoverflow.com/questions/30603309/retrieving-port-number-using-winsock-sockets-api
- https://www.gta.ufrj.br/ensino/eel878/sockets/ | IMPORTANTE