U LISBOA | UNIVERSIDADE DE LISBOA

TP3. Apoio ao projeto 3: sockets

Pedro Ferreira

pmf@ciencias.ulisboa.pt

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

Fonte da imagem: http://www.dsn.jhu.edu

Alguns comandos úteis antes de começar

- netstat -ni
- ifconfig
- ping
- traceroute

Programação com sockets

Objectivo: construir aplicações cliente/servidor que comunicam usando sockets

Socket API

- Introduzido em 1981 para UNIX BSD4.1
- Criado, usado e libertado **explicitamente** pelas aplicações
- Paradigma cliente - servidor
- Dois tipos de serviço de transporte:
  - Com ligação: e.g., TCP
  - Sem ligação: e.g., UDP

socket

Interface *criada pela aplicação mas controlada pelo S.O.* (uma "porta") através da qual os processos podem enviar e receber mensagens de/para outros processos

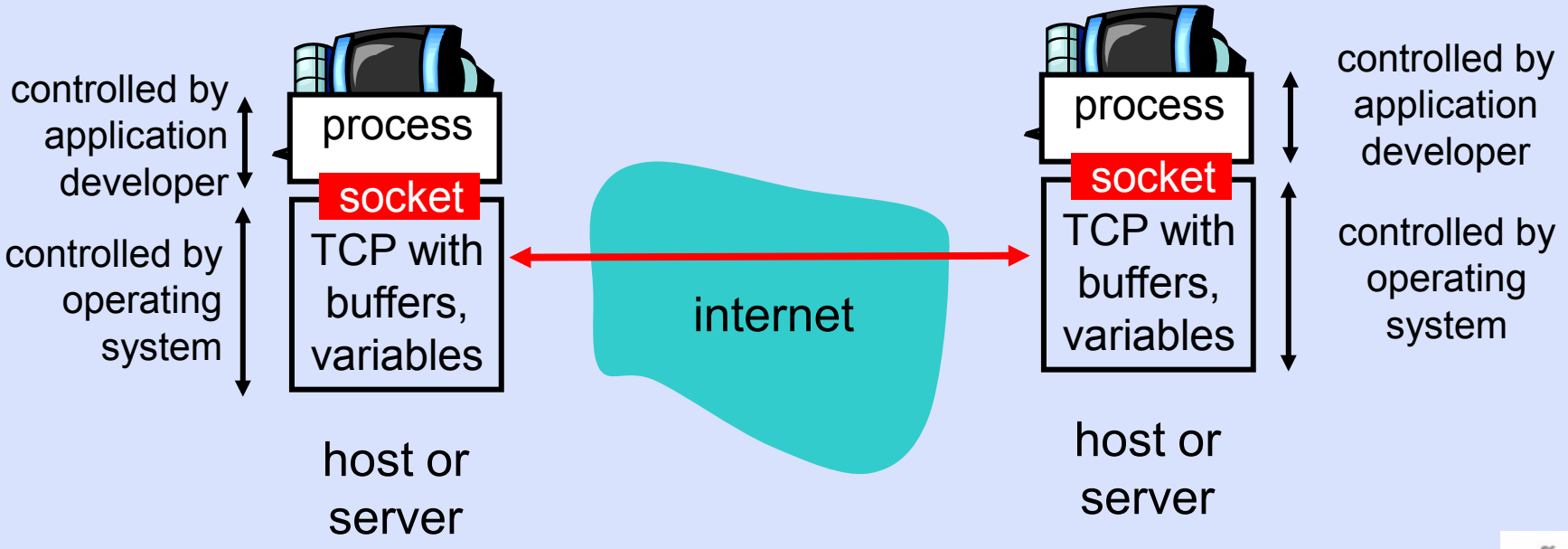
FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Sockets TCP

Socket: porta entre processos

Serviço TCP: : estabelece-se uma ligação antes da comunicação.

Geralmente oferece garantias de fiabilidade e os pacotes são recebidos na ordem em que são enviados (ordem FIFO).



Fonte: [Kurose2009]

Programação de sockets *com TCP*

O cliente deve contactar o servidor

- O processo servidor deve estar já em execução
- O servidor tem de criar um socket (*listening socket*) e ficar à espera do contacto do cliente

Como é que o cliente contacta o servidor?

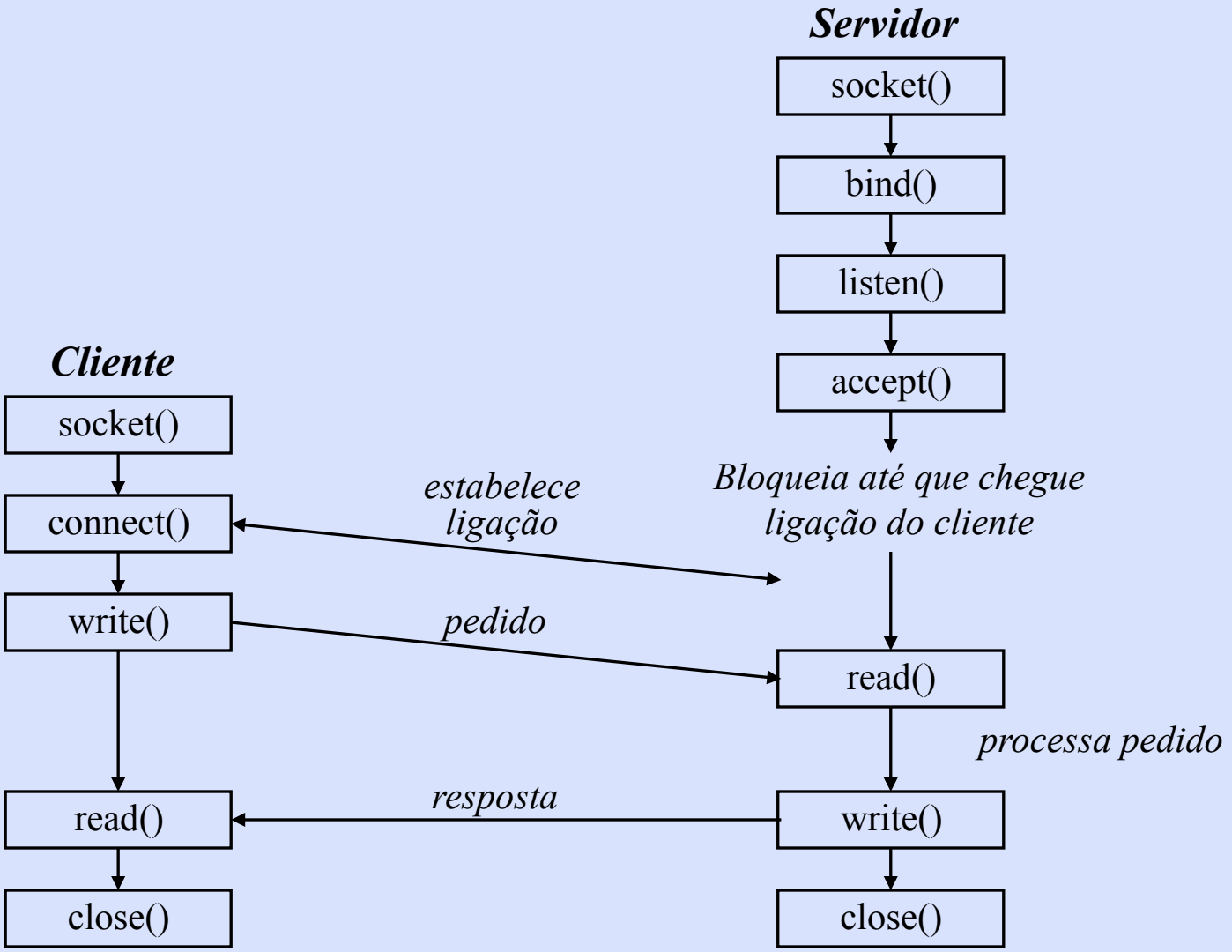
- Cria socket TCP local
  - Especifica um endereço IP e o porto do processo servidor
- Quando o cliente cria o socket estabelece-se ligação TCP entre os dois (*3-way handshake*)

- Quando contactado pelo cliente, o servidor TCP cria novo socket (*connection socket*) para o processo servidor comunicar com o cliente
  - Permitindo que o servidor fale com múltiplos clientes

Do ponto de vista da aplicação:

TCP oferece um serviço fiável, transferindo bytes ordenadamente (como um "pipe") entre cliente e servidor

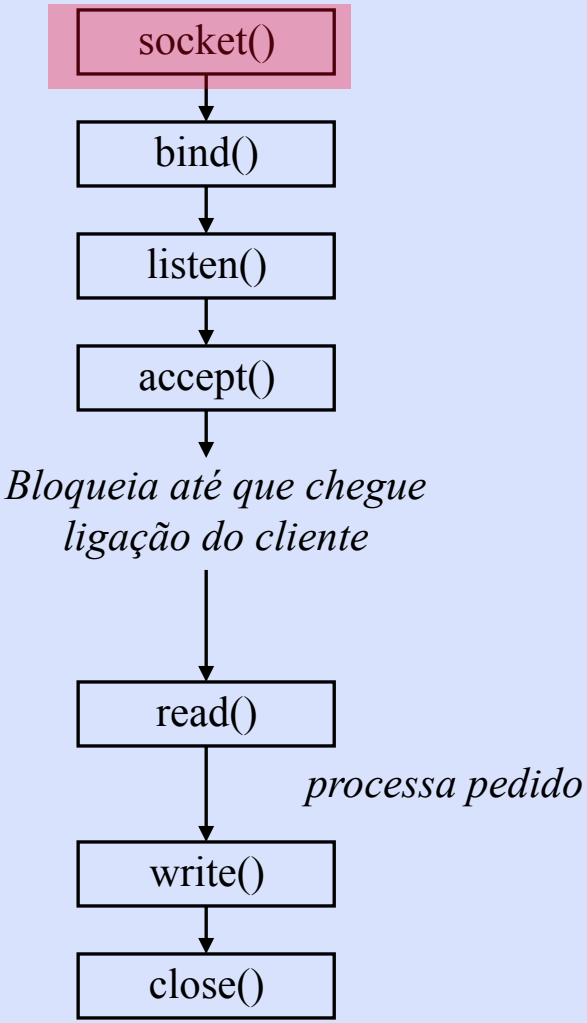
Cliente-servidor com ligação (TCP)



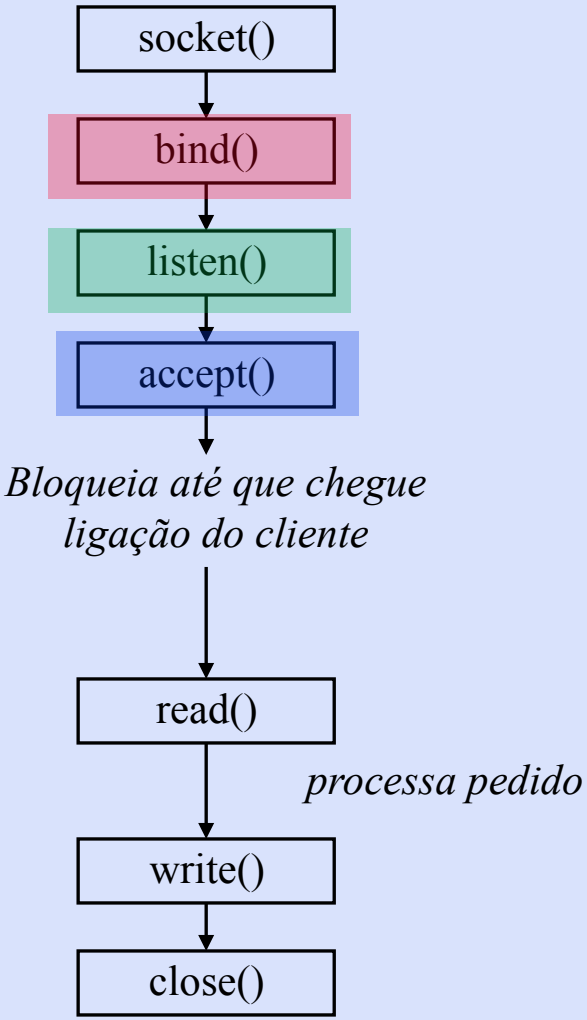
EXEMPLO SERVIDOR TCP

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

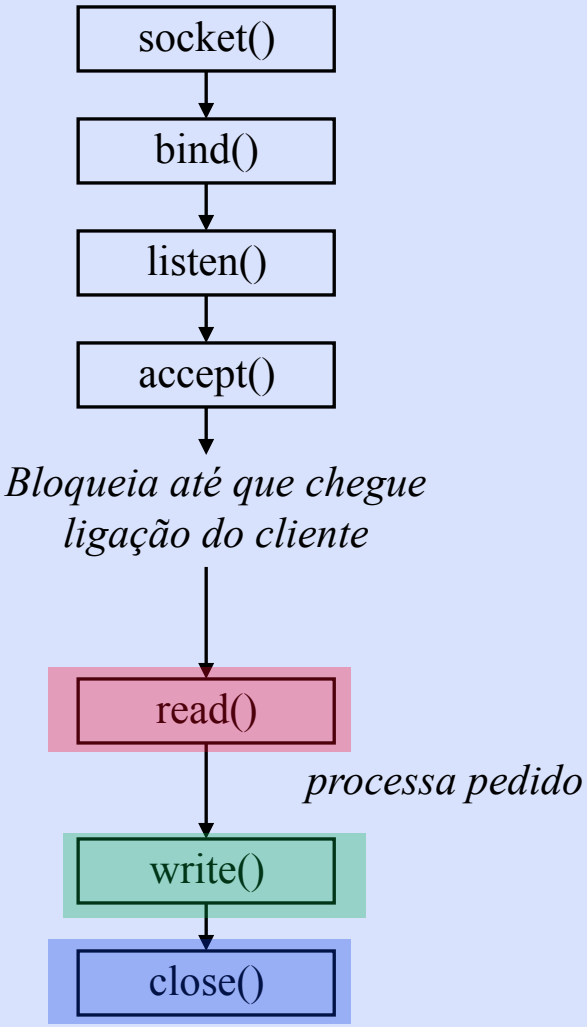
```
/*
 * Programa servidor:
 *   Recebe uma string do cliente e envia o tamanho dessa string.
 *
 * Como compilar:
 *   gcc -Wall -g server.c -o server
 *
 * Como executar:
 *   server <porto_servidor>
 */
#include "inet.h"

int main(int argc, char **argv)
{
   int sockfd, connsockfd;
   struct sockaddr_in server, client;
   char str[MAX_MSG+1];
   int nbytes, count,;
   socklen_t size_client;

   // Verifica se foi passado algum argumento
   if (argc != 2){
      printf("Uso: ./server <porto_servidor>\n");
      printf("Exemplo de uso: ./server 12345\n");
     return -1;
   }
   // Cria socket TCP
   if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
      perror("Erro ao criar socket");
      return -1;
   }
```



```
// Preenche estrutura server para bind
   server.sin_family = AF_INET;
   server.sin_port = htons(atoi(argv[1]));
   server.sin_addr.s_addr = htonl(INADDR_ANY);

   // Faz bind
   if (bind(sockfd, (struct sockaddr *) &server, sizeof(server)) < 0){
      perror("Erro ao fazer bind");
      close(sockfd);
      return -1;
   };

   // Faz listen
   if (listen(sockfd, 0) < 0){
      perror("Erro ao executar listen");
      close(sockfd);
      return -1;
   };

   printf("Servidor 'a espera de dados\n");

// Bloqueia a espera de pedidos de conexão
   while ((connsockfd = accept(sockfd,(struct sockaddr *) &client, &size_client)) != -1) {
```



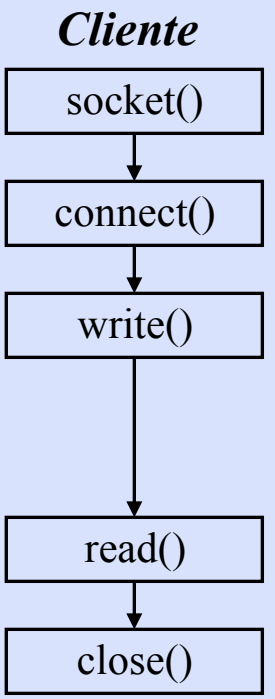
```
// Lê string enviada pelo cliente do socket referente a conexão
    if((nbytes = read(connsockfd,str,MAX_MSG)) < 0){
        perror("Erro ao receber dados do cliente");
        close(connsockfd);
        continue;
    }

    // Coloca terminador de string
    str[nbytes] = '\0';

    // Conta numero de caracteres
    count = strlen(str);

    // Converte count para formato de rede
    count = htonl(count);

    // Envia tamanho da string ao cliente através do socket referente a conexão
    if((nbytes = write(connsockfd,&count,sizeof(count))) != sizeof(count)){
        perror("Erro ao enviar resposta ao cliente");
        close(connsockfd);
        continue;
    }

    // Fecha socket referente a esta conexão
    close(connsockfd);
}

// Fecha socket
close(sockfd);
return 0;
}
```



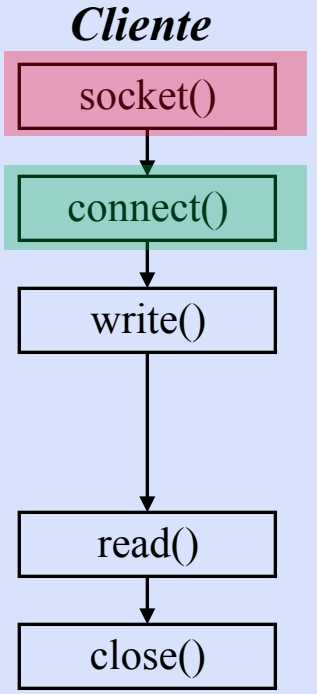
EXEMPLO CLIENTE TCP

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

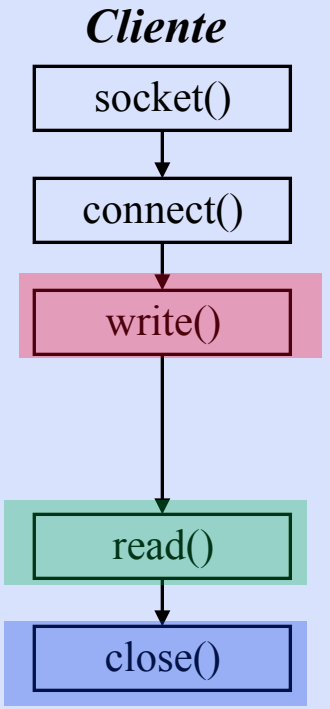
```
/*
 * Programa cliente:
 *   Envia uma string ao servidor, e recebe o tamanho dessa string.
 *
 * Como compilar:
 *   gcc -Wall -g client.c -o client
 *
 * Como executar:
 *   ./client <ip_servidor> <porto_servidor> <string>
 */

#include "inet.h"
#include <errno.h>

int testInput(int argc)
{
   if (argc != 4){
      printf("Uso: ./client <ip_servidor> <porto_servidor> <string>\n");
      printf("Exemplo de uso: ./client 127.0.0.1 12345 olacomovais\n");
      return -1;
   }

   return 0;
}

int main(int argc, char **argv){
   int sockfd;
   struct sockaddr_in server;
   char str[MAX_MSG];
   int count, nbytes;
```



```
// Verifica se foi passado algum argumento
  if (testInput(argc) < 0) return -1;

  // Copia os primeiros MAX_MSG-1 bytes da string passada como argumento
  strncpy(str, argv[3], MAX_MSG-1);

  // Garante que a string tem terminacao
  str[MAX_MSG-1] = '\0';

  // Cria socket TCP
  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
     perror("Erro ao criar socket TCP");
     return -1;
  }

  // Preenche estrutura server para estabelecer conexão
  server.sin_family = AF_INET;
  server.sin_port = htons(atoi(argv[2]));
  if (inet_pton(AF_INET, argv[1], &server.sin_addr) < 1) {
     printf("Erro ao converter IP\n");
     close(sockfd);
     return -1;
  }

  // Estabelece conexão com o servidor definido em server
  if (connect(sockfd,(struct sockaddr *)&server, sizeof(server)) < 0) {
     perror("Erro ao conectar-se ao servidor");
     close(sockfd);
     return -1;
  }
```



```
// Envia string
   if((nbytes = write(sockfd,str,strlen(str))) != strlen(str)){
      perror("Erro ao enviar dados ao servidor");
      close(sockfd);
      return -1;
   }

   printf("'A espera de resposta do servidor ...\n");

   // Recebe tamanho da string
   if((nbytes = read(sockfd,&count,sizeof(count))) != sizeof(count)){
      perror("Erro ao receber dados do servidor");
      close(sockfd);
      return -1;
   };

   printf("O tamanho da string e' %d!\n", ntohl(count));

   // Fecha o socket
   close(sockfd);
   return 0;
}
```



“inet.h”

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

//tamanho maximo da mensagem enviada pelo cliente
#define MAX_MSG 2048
```

EXECUÇÃO

Terminal 1 - Servidor

```
fvramos@fvramos-virtualbox:~/workspace/StringCount$ gcc -Wall -g server.c -o server
fvramos@fvramos-virtualbox:~/workspace/StringCount$ ./server 5000
Servidor 'a espera de dados
```

Terminal 2 - Cliente

```
fvramos@fvramos-virtualbox:~/workspace/StringCount$ gcc -Wall -g client.c -o client
fvramos@fvramos-virtualbox:~/workspace/StringCount$ ./client
Uso: ./client <ip_servidor> <porto_servidor> <string>
Exemplo de uso: ./client 127.0.0.1 12345 olacomovais
fvramos@fvramos-virtualbox:~/workspace/StringCount$ ./client 127.0.0.1 4000 Ola
Erro ao conectar-se ao servidor: Connection refused
fvramos@fvramos-virtualbox:~/workspace/StringCount$ ./client 127.0.0.1 5000 Ola
'A espera de resposta do servidor ...
O tamanho da string e' 3!
```

Especificação de endereços: Internet

- ***struct in_addr***: estrutura com um endereço Internet (4 bytes)

```
#include<netinet/in.h>

struct in_addr {
  uint32_t s_addr;      /* Endereço IP, formato rede */
};
```

- ***struct sockaddr_in***: endereço para a família de protocolos Internet

exemplo

```
server.sin_family = AF_INET;
server.sin_port = htons(atoi(argv[2]));
if (inet_pton(AF_INET, argv[1], &server.sin_addr) < 1) {
```

```
#include<netinet/in.h>

struct sockaddr_in {
  uint8_t         sin_len;     /* Compr. Não existe no Linux */
  sa_family_t     sin_family;  /* AF_INET */
  in_port_t       sin_port;    /* porto, formato rede */
  struct in_addr  sin_addr;    /* endereço IP, formato rede */
  char            sin_zero[8]; /* Não usado */
};
```

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Outras especificação dos endereços para sockets

- ***struct sockaddr***: estrutura genérica para todas as famílias protocolos

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

struct sockaddr {
  uint8_t sa_len;              /* Comprimento (opcional) */
  sa_family_t sa_family;       /* AF_XXX */
  char sa_data[14];            /* Específica por protocolo
  */
};
```

exemplo

*if (connect(sockfd,(struct sockaddr *)&server, sizeof(server)) < 0)*

- ***struct sockaddr_un***: endereço para a família de protocolos domínio Unix

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

struct sockaddr_un {
  uint8_t     sun_len;       /* Compr. Não existe no Linux
  */
  sa_family_t sun_family;     /* AF_LOCAL */
  char        sun_path[104]; /* endereço Unix é um
  pathname */
};
```


Especificação de Endereços: Internet/IPv6

- ***struct in6_addr*** : estrutura com um endereço Internet para IPv6

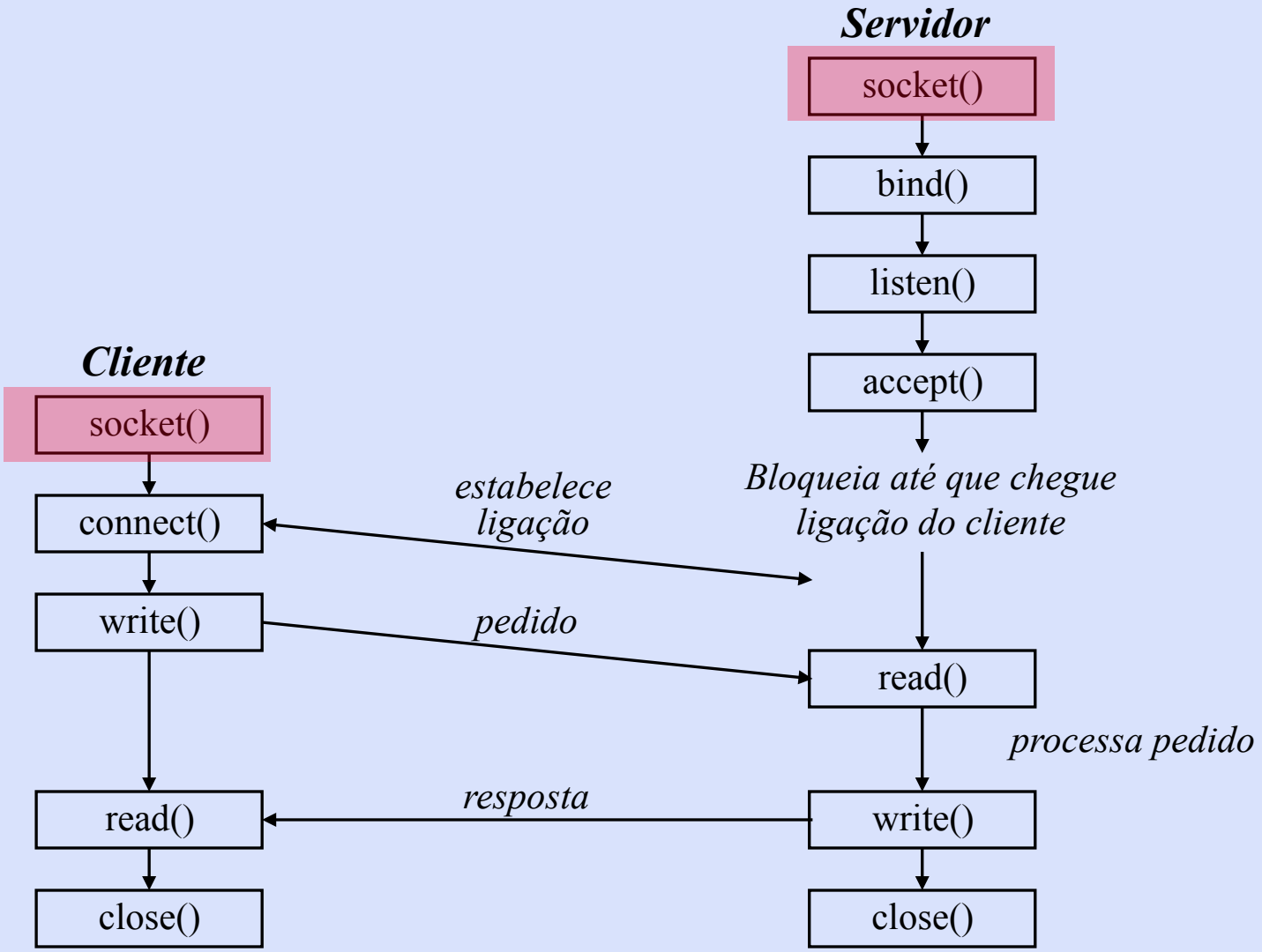
```
#include<netinet/in.h>

struct in6_addr {
  uint8_t s6_addr[16];   /* Endereço IPv&6 de 128 bits */
                         /* em network byte order */
};
```

- ***struct sockaddr_in6*** : endereço para a família de protocolos Internet para IPv6

```
#include<netinet/in.h>

struct sockaddr_in6 {
    uint8_t sin6_len;
    sa_family_t sin6_family;    /* AF_INET6 */
    in_port_t sin6_port;        /* Transport layer port # */
    uint32_t sin6_flowinfo;     /* IPv6 flow information */
    struct in6_addr sin6_addr;  /* IPv6 address */
};
```

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Função *socket()*



Função *socket()* (cont.)

- ***socket()*** : cria uma socket e retorna um descritor para o socket

exemplo

if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {

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

int socket(int family, int type, int protocol);
```

Famílias de protocolos *(AF - Address Family)*:

```
AF_INET          protocolos IPv4
AF_INET6         protocolos IPv6
AF_LOCAL         protocolos do domínio Unix (AF_UNIX no
  passado)
AF_ROUTE         protocolos de encaminhamento
AF_KEY           socket key
```

Tipo:

```
SOCK_STREAM      socket com-ligação, stream de bytes
SOCK_DGRAMsocket sem-ligação
SOCK_RAW         socket raw
```


Função *socket()* (cont.)

Combinações válidas:

	AF_INET	AF_INET6	AF_LOCAL	AF_ROUTE	AF_KEY
SOCK_STREAM	TCP	TCP	Sim		
SOCK_DGRAM	UDP	UDP	Sim		
SOCK_RAW	IPv4	IPv6		Sim	Sim

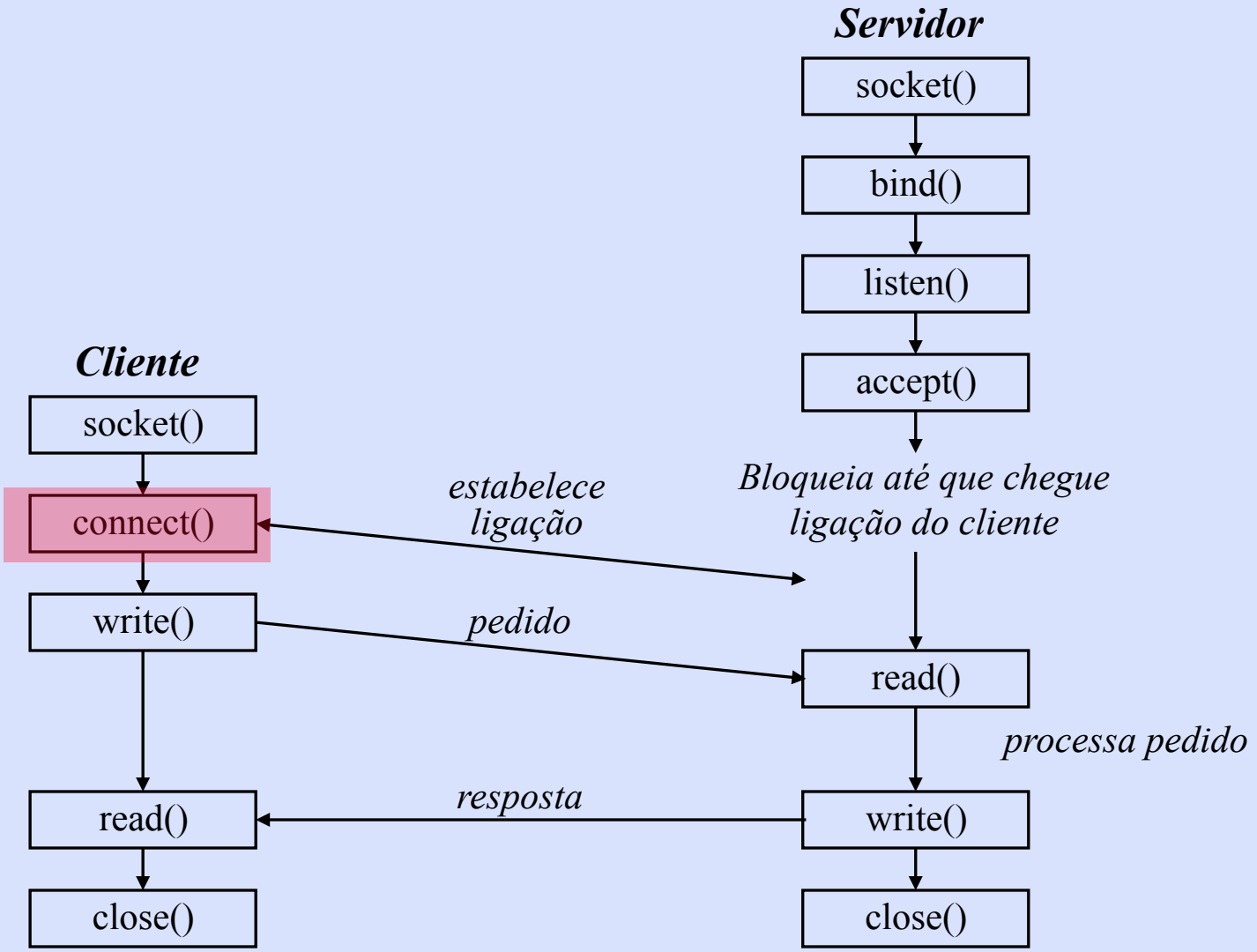
Protocolo: tipicamente colocado a 0.

```
#include <netinet/in.h>
```

Família	Tipo	Protocolo	Designação
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMPv4
AF_INET	SOCK_RAW	IPPROTO_RAW	IP
AF_INET6	SOCK_RAW	IPPROTO_ICMP6	ICMPv6

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Função *connect()*



Função *connect()* (cont.)

- ***connect()*** : pedido de estabelecimento de uma ligação (cliente).

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

int connect(int sockfd, struct sockaddr *serv_addr,
            socklen_t addrlen);
```

```
typedef unsigned int socklen_t;
```

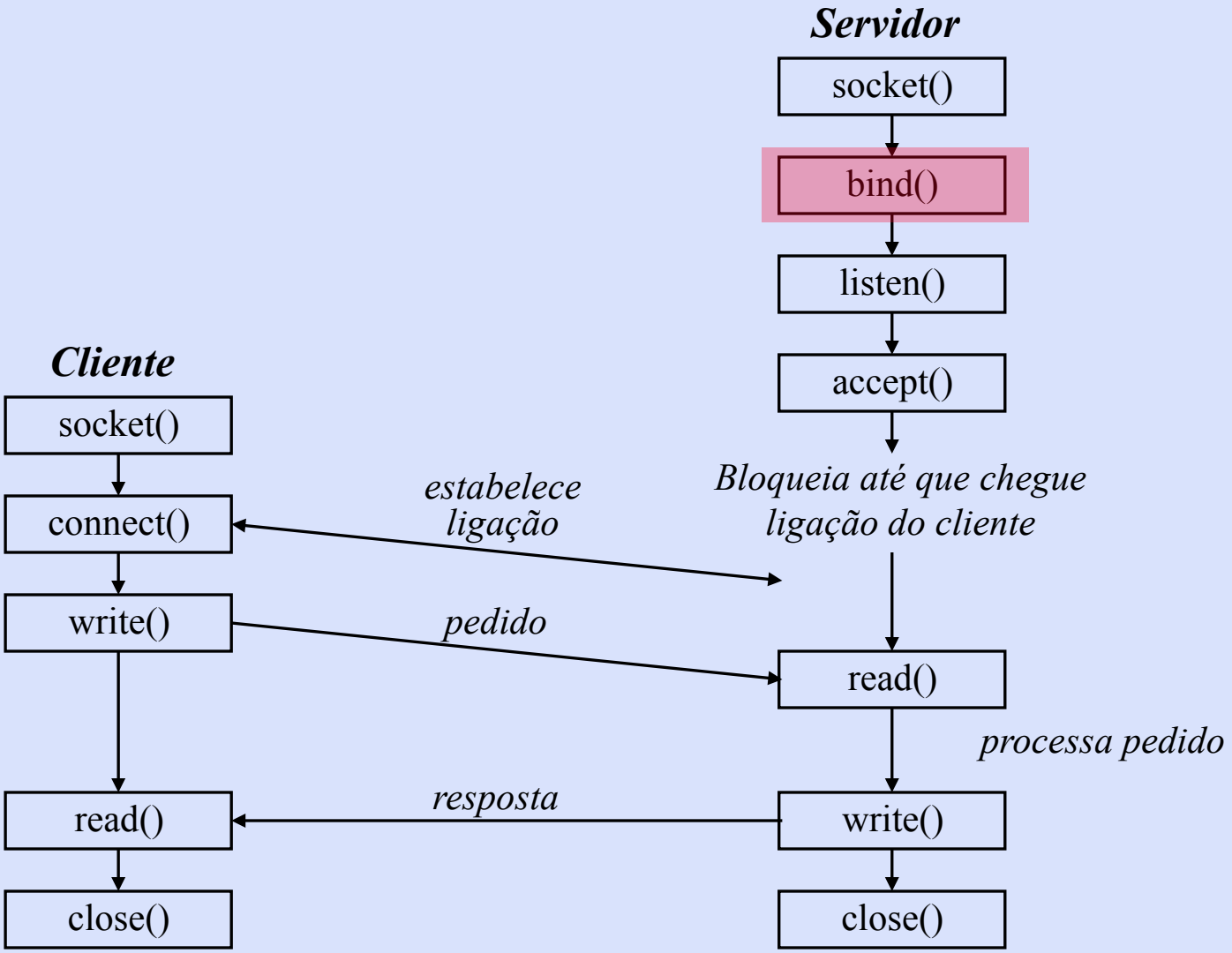
- Devolve: 0 se OK, -1 em caso de erro

exemplo

*if (connect(sockfd,(struct sockaddr *)&server, sizeof(server)) < 0)*

- Notas :
  - num protocolo com ligação, o cliente não precisa de chamar *bind()* antes de fazer *connect()* porque o *connect()* atribuí-lhe o endereço local automaticamente

Função *bind()*



Função *bind()* (cont.)

- ***bind()*:** associa um endereço local a um *socket*

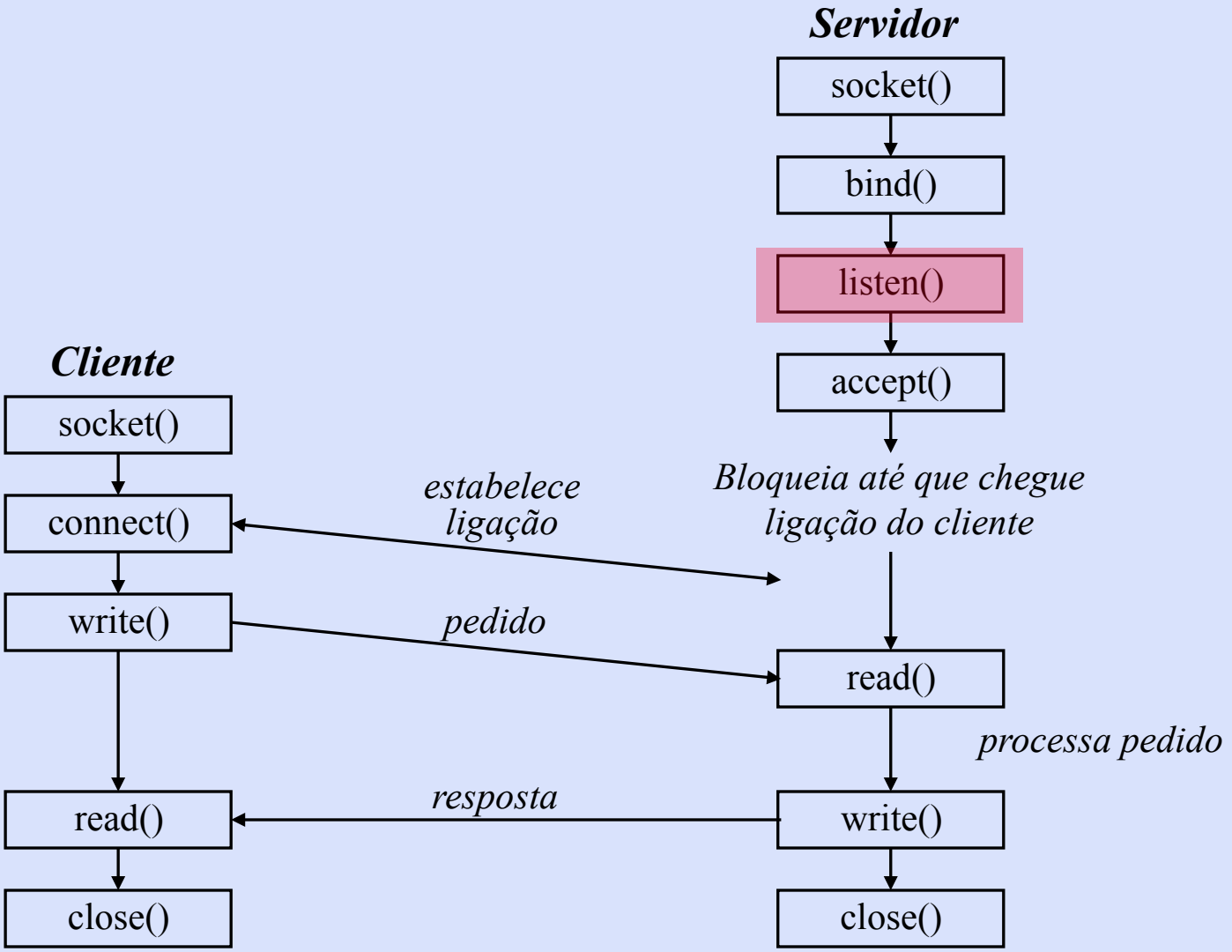
```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr *my_addr,
         socklen_t addrlen);
```

- Devolve: 0 se OK, -1 em caso de erro

exemplo

```
server.sin_addr.s_addr = htonl(INADDR_ANY);

...

if (bind(sockfd, (struct sockaddr *) &server, sizeof(server)) < 0)
```


Função *listen()*



Função *listen()* (cont.)

- ***listen()*** : indica a intenção de aceitar ligações e o número máximo de ligações em espera.

exemplo

if (listen(sockfd, 0) < 0)

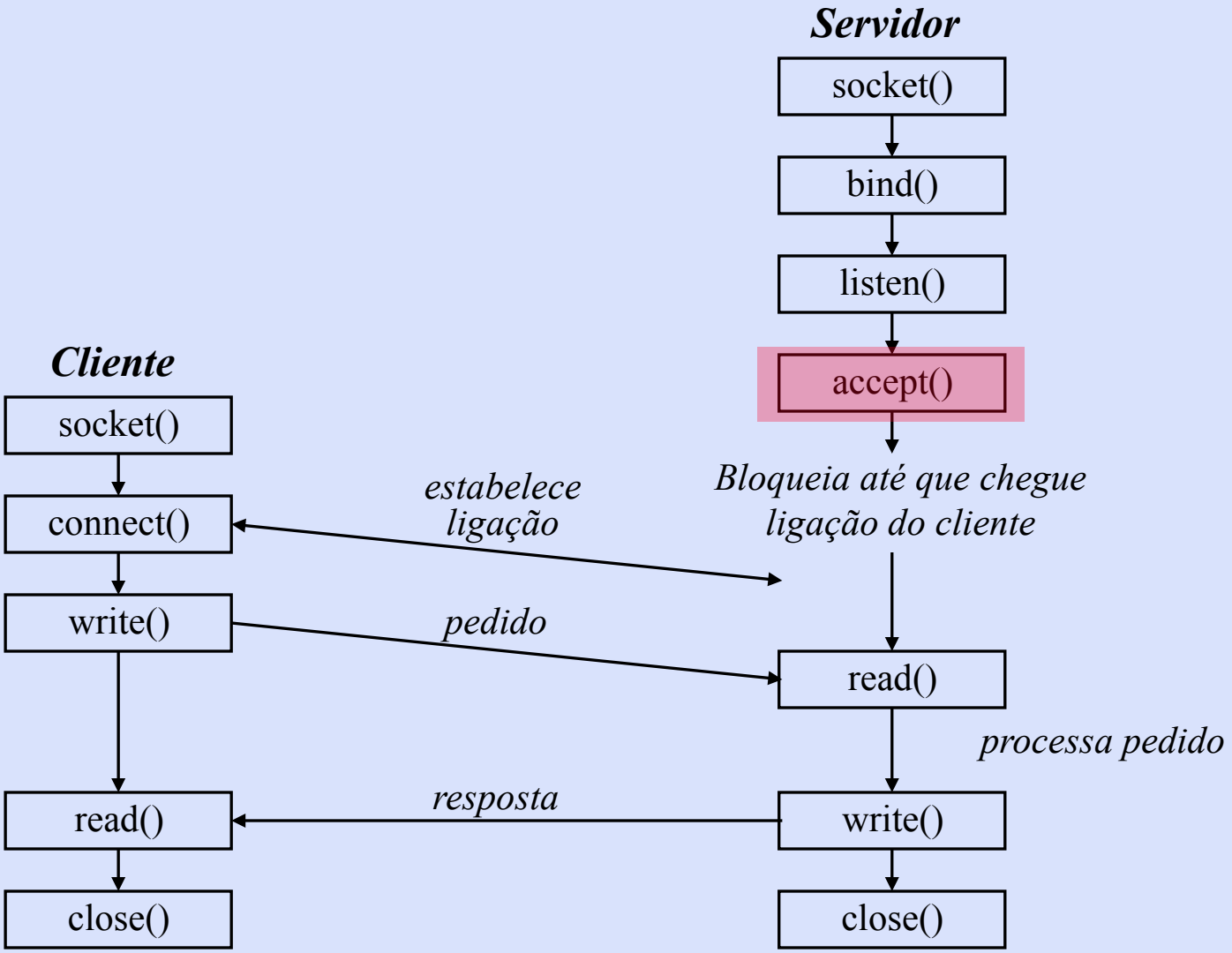
```
#include <sys/socket.h>

int listen(int sockfd, int backlog);
```

`backlog`: nº máximo de ligações pendentes. Dependendo da implementação pode ser o número de ligações para as quais ainda não terminou o *three-way handshake* do TCP iniciado por um cliente, ou o número de ligações estabelecidas para as quais não se fez `accept()`.

  - » limitado por SO_MAXCONN (128 em Linux, 5 em BSD)

- Devolve: 0 se OK, -1 em caso de erro

Função *accept()*



Função *accept()* (cont.)

- ***accept()*** : aceita uma das ligações em espera e cria uma nova socket com as mesmas características que *sockfd*. Se não existir nenhuma ligação pendente, o processo fica bloqueado.

exemplo

```
while ((connsockfd = accept(sockfd,(struct sockaddr *) &client, &size_client)) != -1) {
```

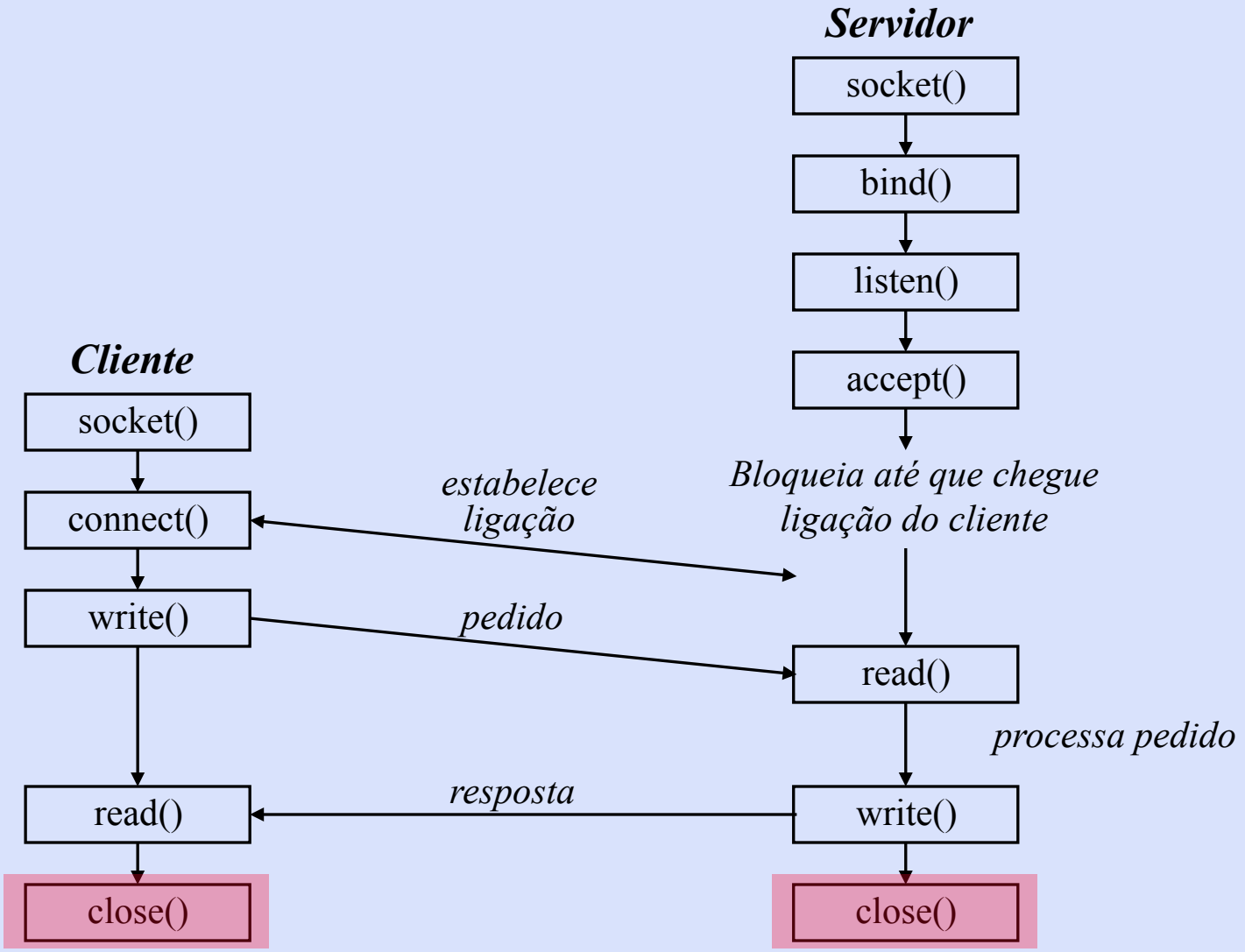
```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *addr, socklen_t
  *addrlen);
```

`addr` : endereço do interlocutor

`addrlen`: iniciado com o tamanho de `addr`, e retorna o tamanho do endereço

- Devolve: o descritor da nova socket ou -1 em caso de erro

Função *close()*



Funções *close() / shutdown()* (cont.)

- ***close()*** : fecha um socket. O S.O. nos protocolos com garantias de fiabilidade tenta enviar os dados que ainda não tenham sido transmitidos ou *ack*.

```
#include <unistd.h>

int close(int sockfd);
```

exemplo

close(sockfd);

- ***shutdown()*** : maior controlo ao fechar o socket

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

int shutdown(int sockfd, int howto);

howto:
   SHUT_RD      (0): impossibilita recepção
   SHUT_WR      (1): impossibilita envio
   SHUT_RDWR    (2): impossibilita envio e recepção
```

- Devolvem: 0 se OK, -1 em caso de erro

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Resumo da criação de uma associação

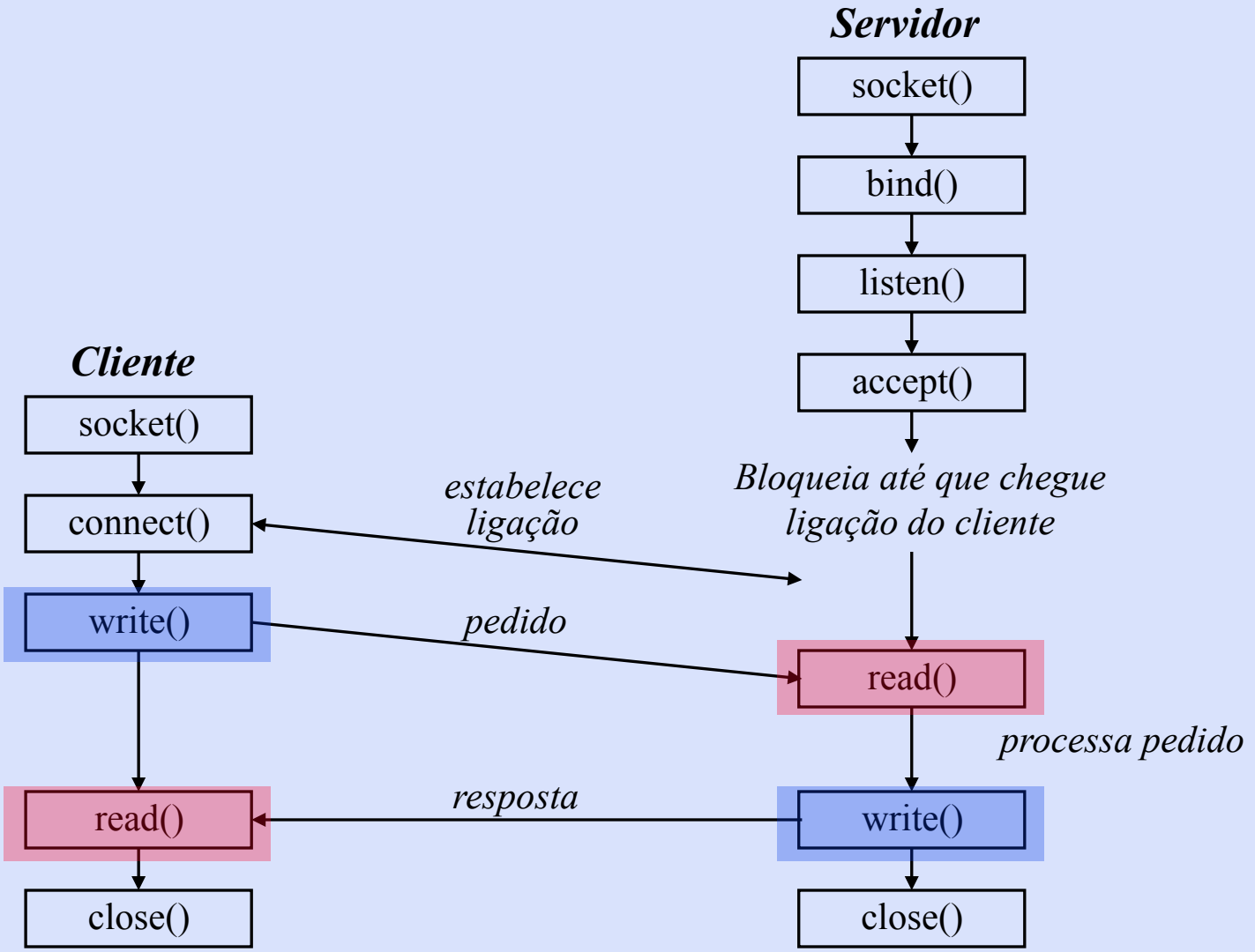
- **Associação**

  {protocolo, endereço-local, proc-id-local, endereço-remoto, proc-id-remoto}

	protocolo	endereço-local & proc-id-local	endereço-remoto & proc-id-remoto
servidor com-ligação	*socket()*	*bind()*	*listen(), accept()*
cliente com-ligação	*socket()*	*connect()*	
servidor sem-ligação	*socket()*	*bind()*	*recvfrom()*
cliente sem-ligação	*socket()*	*bind()*	*sendto()*

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Funções *read()* e *write()*



Funções *read() / write ()*

- As operações de leitura e escrita em *sockets* podem também ser realizadas usando as funções padrão da API do UNIX - *read* e *write* – da mesma forma que estas são usadas com ficheiros

- ***read() / write()*** : leitura e escrita com *buffers* contínuos

```
#include <unistd.h>

ssize_t read(int fd, const void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

exemplo

```
if((nbytes = write(sockfd,str,strlen(str))) != strlen(str)){
...
if((nbytes = read(sockfd,&count,sizeof(count))) != sizeof(count)){
```

- Devolvem: número de bytes lidos ou escritos ou -1 em caso de erro.

- Obs: `size_t` e `ssize_t` são inteiros

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Funções *send() / recv()*

- ***send() / recv()*** : envio ou recepção de mensagens, normalmente para sockets com ligação. Bloqueiam até que o *buf* de emissão tenha espaço livre (*send*) ou até terem sido recebidos dados (*recv*).

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

int send(int s, void *buf, int len, int flags);
int recv(int s, void *buf, int len, int flags);
```

`flags` : iguais a zero ou o *or* destas (e de outras) constantes:

MSG_OOB : dados urgentes (canais com ligação)

MSG_PEEK : lê uns dados sem os retirar da fila (*recv*)

MSG_DONTROUTE : evita o encaminhamento normal (*send*)

- Devolve: o número de bytes lidos/recebidos ou -1 em caso de erro

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Funções *sendto() / recvfrom()*

- ***sendto() / recvfrom()*** : Envio ou recepção de mensagens com indicação do destinatário ou emissor. Bloqueiam até que o *buf* de emissão tenha espaço livre (*send*) ou até terem sido recebidos dados (*recv*).

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

`size_t` : 32 bits ou 64 bits, conforme a arquitectura da máquina

```
int sendto(int sockfd, void *msg, size_t len, int flags,
           struct sockaddr *to, socklen_t tolen);

int recvfrom(int sockfd, void *buf, size_t len, int flags,
             struct sockaddr *from, socklen_t *fromlen);
```

- Devolve: o número de bytes enviados/recebidos ou -1 em caso de erro

Funções *sendto() / recvfrom()* (cont.)

Argumentos:

`to` : indica o destinatário da mensagem

`from`: endereço do emissor ou pode ser NULL

`tolen/fromlen`: tamanho dos endereços (em que o `fromlen` começa por ter o tamanho da estrutura `from`, e depois vem com o valor que foi realmente preenchido)

`flags` : iguais a zero ou o *bitwise or* das seguintes constantes:

MSG_PEEK: lê uns dados sem os retirar da fila (*recvfrom*)

MSG_DONTROUTE: evita os mecanismos normais de encaminhamento (*sendto*)

Resumo das Funções de Escrita/Leitura

Ver man pages

Operação	*Todos tipos de descrit.*	*Só de sockets*	*Só um buffer*	*Scatter / gather*	*Tem flags*	*Tem endereço da outra parte*	*Tem direitos de acesso*
read, write	*		*				
readv, writev	*			*			
recv, send		*	*		*		
recvfrom, sendto		*	*		*	*	
recvmsg, sendmsg		*		*	*	*	*

Ver man pages

Portos

- Reserva de portos
  - *pelo processo*: o processo especifica um endereço com um dado porto (usado tipicamente por servidores que querem associar uma socket a um endereço bem conhecido)
  - *pelo sistema*: se o processo não indica o porto (quando faz ***bind()*** coloca o porto a 0) o sistema automaticamente associa um porto à socket

Reservados : 1-1023

Dinâmicos: 49152-65535

Livres : o resto ☺

ftp:	21/tcp
telnet:	23/tcp
smtp:	25/tcp
domain(DNS):	53/tcp/udp
http:	80/tcp
nntp:	119/tcp

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Funções de conversão de dados

- ***htonl() / htons()*** : converte do formato da máquina um long/short para o formato standard da rede
- ***ntohl() / ntohs()*** : converte de formato standard da rede um long/short para o formato da máquina

exemplo

```
server.sin_port = htons(atoi(argv[1]));
server.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
#include <netinet/in.h>

uint16_t htons(uint16_t host16bitvalue);
uint32_t htonl(uint32_t host32bitvalue);
```

- Devolvem: valor no formato da rede

```
uint16_t ntohs(uint16_t host16bitvalue);
uint32_t ntohl(uint32_t host32bitvalue);
```

- Devolvem: valor no formato do computador

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Funções de conversão de endereços

- Funções ***inet_pton()*** e ***inet_ntop()***
  - Funcionam, quer para o IPv4, quer para o IPv6
  - `inet_pton()` converte de uma *string* (a.b.c.d no IPv4 ou x:x:x:x:x:x:a.b.c.d ou x:x:x:x:x:x:x:x no IPv6) para um endereço na estrutura `struct in_addr` ou `struct in6_addr`, conforme o caso.
  - `inet_ntop()` faz o contrário

exemplo

```
if (inet_pton(AF_INET, argv[1], &server.sin_addr) < 1) {
```

```
int inet_pton(int family, const char *strptr, void *addrptr);
```

- Devolve: 1 se OK, 0 se a entrada não é um formato de apresentação válida, -1 em caso de erro

```
char *inet_ntop(int family, const void *addrptr, char *strptr,
                size_t len);
```

- Devolve: ponteiro para o resultado se OK, NULL em caso de erro

Erros

- (Quase) todas as funções podem falhar
  - É importante verificar todos os erros possíveis
  - Típicamente, devolvem -1 (int) para indicar um erro (ou ponteiro NULL em alguns casos)

- Em caso de erro:
  - `errno`: inteiro global com código de erro
    - » Cada thread tem o seu `errno` (valor de retorno das funções Pthread)
  - `perror()`: função para imprimir descrição textual do erro
  - Sem erro, o valor de `errno` é indefinido

- `errno.h` define constantes para os códigos de erro

- Erros "especiais" não indicam erro: o **programa pode (e deve) repetir a função**:
  - EINTR (se usar sinais)
  - EWOULDBLOCK (em operações de I/O não bloqueante)

Tipos de erros relacionados com sockets

- `EACCES`

  Permissão para criar socket desse tipo negada.

- `EAFNOSUPPORT`

  A implementação não suporta essa família de endereços

- `EINVAL`

  Passagem de argumento inválido para uma função.

- `EMFILE`

  Já estão muitos ficheiros abertos no processo. Não se podem abrir mais.

- `ENFILE`

  Já estão muitos ficheiros abertos no sistema. Não se podem abrir mais.

- `ENOBUFS` ou `ENOMEM`

  Memória insuficiente. Socket não pode ser criado.

- `EPROTONOSUPPORT`

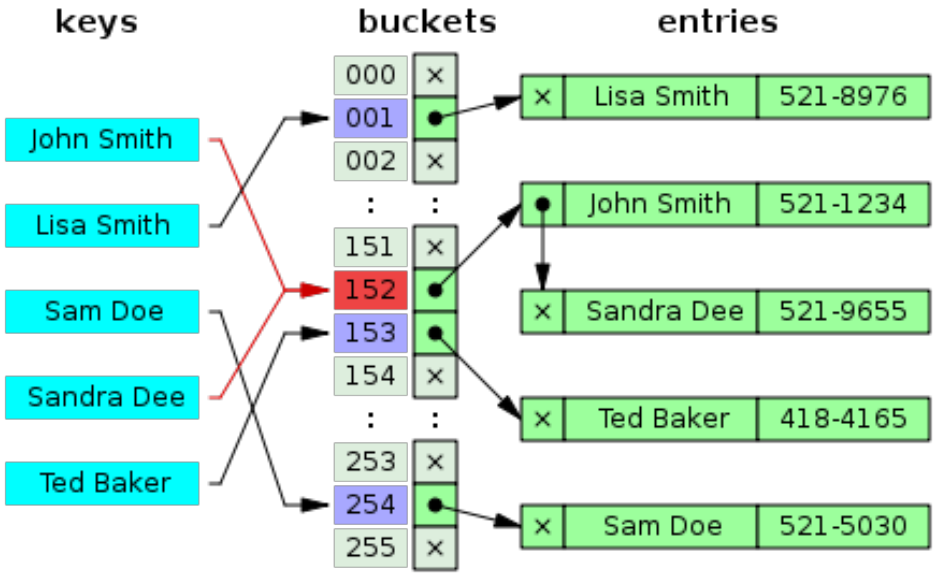
  O protocolo não é suportado.

Exemplo

```
int write_all(int sock, char *buf, int len) {
    int bufsize = len;
    while(len>0) {
        int res = write(sock, buf, len);
        if(res<0) {
            if(errno==EINTR) continue;
            perror("write failed:");
            return res;
        }
        buf += res;
        len -= res;
    }
    return bufsize;
}
```


Revisitando o projeto 1 e 2: tabela *hash* com *chaining*

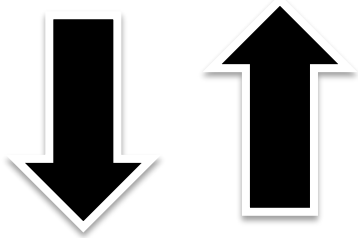
- Objectivo: Espaço de tuplos através de tabela hash com chining. (Armazenamento de pares chave-valor similar ao utilizado pela Amazon [DeCandia2007])
  - Estrutura de dados: **tabela *hash* com chaining**.



Fonte: http://en.wikipedia.org/wiki/Hash_table

Revisitando o projeto 2: *marshalling* e *unmarshalling*

```
struct message_t {
        short opcode; /* codigo da operação na mensagem */
        short c_type; /* tipo do content da mensagem */
        union content_u {
                struct entry_t *entry;
                char *key;
                char **keys;
                struct data_t *value;
                int result;
        } content; /* conteúdo da mensagem */
};
```



message_to_buffer()

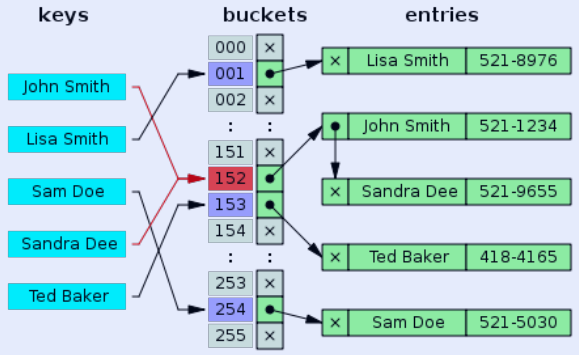
buffer_to_message()

Projeto 3: aplicação cliente-servidor

CLIENTE

```
OUT AB CD EF
IN A * *
COPY_ALL * * *
IN_ALL A * *
Size
quit
```

SERVIDOR



FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

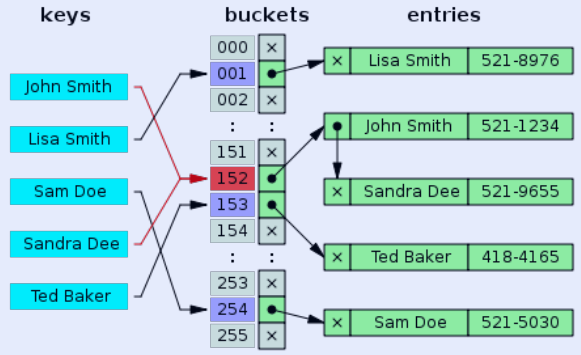
Projeto 3: aplicação cliente-servidor

CLIENTE

"10 100 3 AB CD EF"

```
OUT AB CD EF
IN A * *
COPY_ALL * * *
IN_ALL A * *
Size
quit
```

SERVIDOR



FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

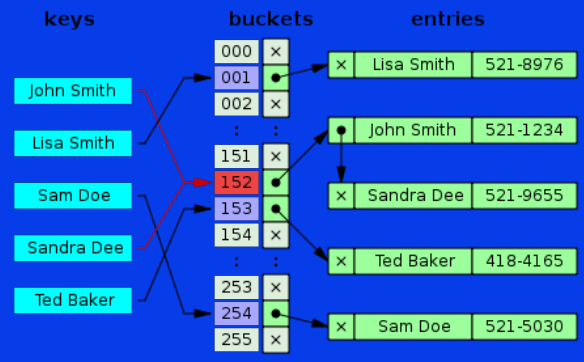
Projeto 3: aplicação cliente-servidor

CLIENTE

```
OUT AB CD EF
IN A * *
COPY_ALL * * *
IN_ALL A * *
Size
quit
```

SERVIDOR

"10 100 3 AB CD EF"



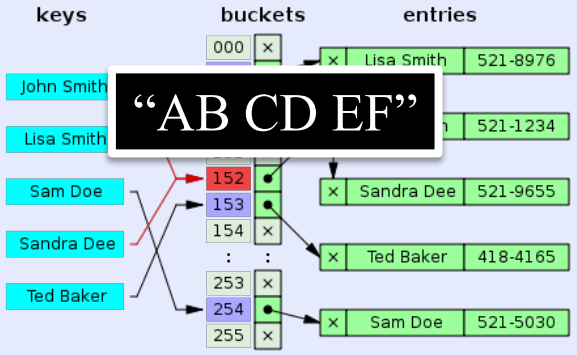
FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Projeto 3: aplicação cliente-servidor

CLIENTE

```
OUT AB CD EF
IN A * *
COPY_ALL * * *
IN_ALL A * *
Size
quit
```

SERVIDOR



FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Projeto 3: aplicação cliente-servidor

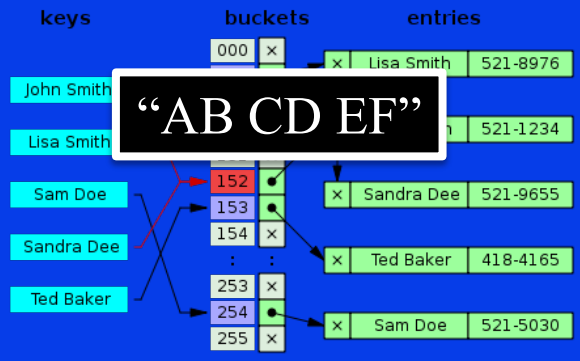
CLIENTE

```
OUT AB CD EF
IN A * *
COPY_ALL * * *
IN_ALL A * *
Size
quit
```

“11 300 2”

SERVIDOR

“AB CD EF”



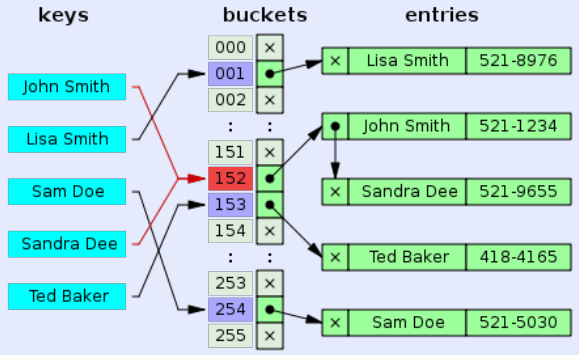
Projeto 3: aplicação cliente-servidor

CLIENTE

"11 300 2"

```
OUT AB CD EF
IN A * *
COPY_ALL * * *
IN_ALL A * *
Size
quit
```

SERVIDOR



FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Definição do OPCODE e C_TYPE

COMANDO CLIENTE	OPCODE PEDIDO	OPCODE RESPOSTA	C_TYPE PEDIDO	CT_TYPE RESPOSTA
OUT	OC_OUT	OC_OUT+1	CT_TUPLE	CT_RESULT
IN	OC_IN	OC_IN+1	CT_TUPLE	CT_RESULT
IN_ALL	OC_IN	OC_IN_ALL+1	CT_TUPLE	CT_RESULT
COPY	OC_COPY	OC_COPY+1	CT_TUPLE	CT_RESULT
COPY_ALL	OC_COPY_ALL	OC_COPY_ALL+1	CT_TUPLE	CT_RESULT
SIZE	OC_SIZE	OCSIZE+1	-	CT_RESULT
quit	-	-	-	-

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Cliente em mais detalhe…

`table_client.c`

main()

`network_client.c`

network_connect()
network_send_receive()
network_close()

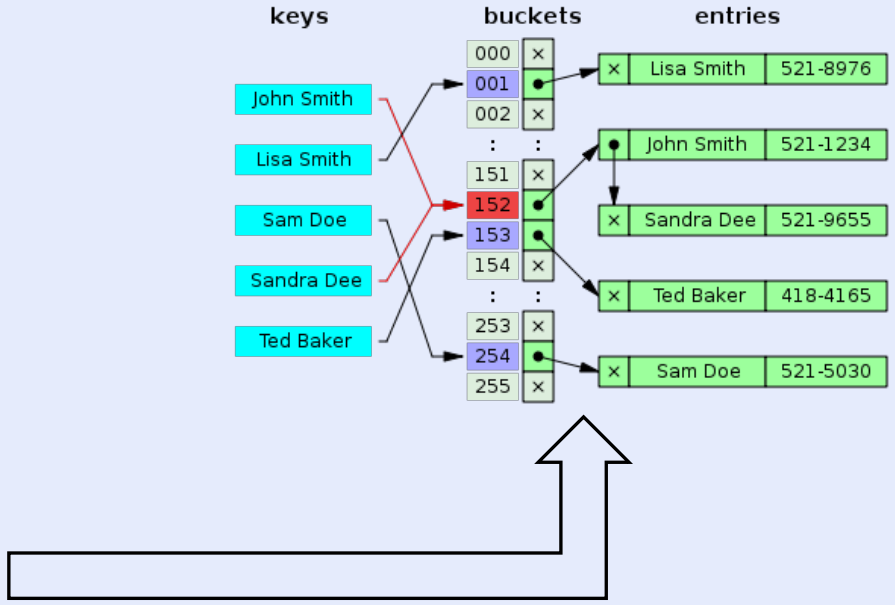
Interface com o utilizador, construção de `struct message_t` para enviar ao módulo de comunicação

Serializa mensagem (usando `message_to_buffer()`), envia ao servidor, espera por resposta.

Servidor em mais detalhe…

`table_server.c`

main()



Recebe pedidos, cria `struct message_t`, iterpreta `opcode` da `struct message_t` e executa operação na tabela. Envia resultado nessa mesma estrutura. Depois, serializa mensagem e envia resposta

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Notas finais: comunicação

- Antes de enviarem mensagens ao interlocutor, servidor e cliente devem enviar primeiro uma outra mensagem curta contendo um inteiro (4 bytes) em formato de rede indicando o tamanho da mensagem principal que vão enviar de seguida.
  - Isto ajuda a garantir que se recebem os dados todos.
- Quando o servidor tem de enviar o resultado de uma pesquisa, fá-lo da seguinte forma:
  - Se o resultado for um conjunto vazio, o servidor:
    - » Forma uma mensagem com {OPCODE pedido +1, CT_RESULT, 0}
  - Se o resultado for um conjunto de **n** tuplos, o servidor:
    - » Forma uma mensagem com {OPCODE pedido +1, CT_RESULT, **n**}
  - Envia 4 bytes com tamanho da mensagem no formato rede (htonl)
  - De seguida envia a mensagem formada e no caso da pesquisa vazia, terminou.
  - Para cada tuplo, forma a mensagem com o tuplo, determina o seu tamanho, envia mensagem curta com o tamanho e depois a mensagem.

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

Notas finais: comunicação

- Recomenda-se a criação de funções `read_all` e `write_all` que vão receber e enviar strings inteiras pela rede
  - Ver exemplo anterior e figuras 3.15 e 3.16 de [Stevens2004]

- Usar a função signal() para ignorar sinais do tipo SIGPIPE. Isto deve ser feito para evitar que um programa morra quando a outra parte é desligada.

- Em caso de erro retornar {OP_ERROR, CT_RESULT, *errcode*}

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Exercício: troca de palavras secretas

- Dividir a turma em duas partes
  - metade esquerda desenvolve o **servidor**
  - metade direita desenvolve o **cliente**

- Cliente envia **palavra secreta A** ao servidor (pede palavra ao utilizador)
- Servidor lê palavra secreta A, imprime-a no ecrã e envia nova **palavra secreta B** de resposta (pede palavra ao utilizador)
- Cliente lê palavra secreta B e imprime-a no ecrã
- **Condição de sucesso**: servidor e cliente apresentam no ecrã, respetivamente, a palavra secreta A e B

- ***Opcional:*** *Antes de enviarem as palavras secretas, o cliente/servidor pode enviar primeiro uma outra mensagem curta contendo um inteiro (4 bytes) em formato de rede indicando o tamanho da palavra que vai ser enviada a seguir.*

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Referências

- [Stevens2004]
  - W. R. Stevens, B. Fenner, A.M. Rudoff, *Unix Network Programming, The Sockets Networking API,* Volume 1, 3rd Edition, Addison Wesley, 2004
- [Kurose2009]
  - J. Kurose, K. Ross , “Computer Networking: A Top Down Approach “,5th edition, Addison-Wesley, April 2009.