Computação Distribuída

Odorico Machado Mendizabal

Universidade Federal de Santa Catarina – UFSC
Departamento de Informática e Estatística – INE

Sockets

Primitivas de Comunicação – *Sockets*

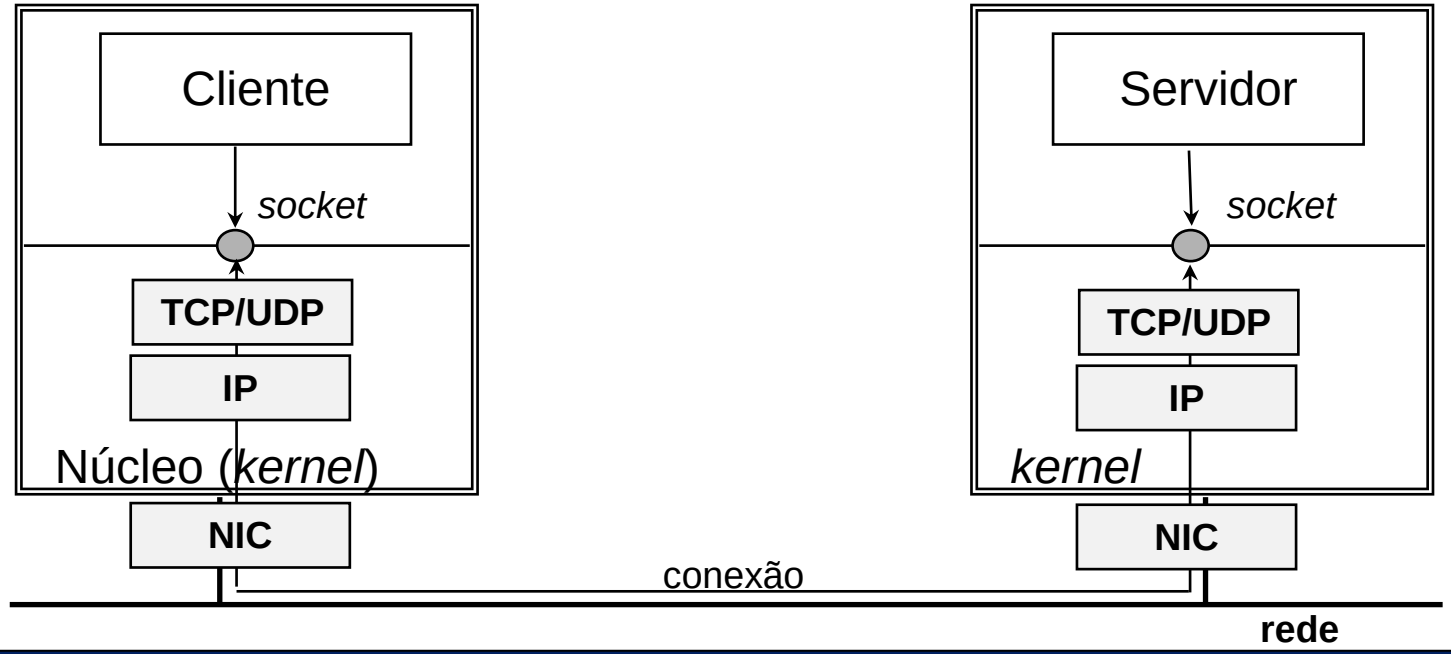
- Propostas inicialmente no Unix, atualmente na maioria dos sistemas operacionais
  - São uma extensão do conceito de *pipes (Berkeley Unix)*

- Chamadas de sistema implementadas sobre a camada de transporte (ex. TCP e UDP)

- Endereço de *socket*
  - Identificador de comunicação formado por um endereço de rede (ex. IP) e um identificador de porta local

Comunicação usando *Sockets* – Visão Geral

• Operações de comunicação entre processos usam pares de *sockets*, 1 em cada lado da comunicação
• Mensagens são transferidas para um *buffer* do *socket* de envio, transmitidas para o socket destino.
  - Dependendo do tipo de comunicação, mensagens podem ser entregues na tentativa de melhor esforço ou enfileiradas em *buffer* no destinatário até que o destinatário consuma as mensagens



Atributos de um *Socket*

- *Sockets* são caracterizados por 3 atributos:
  - Domínio / Família de Endereço
    - *Ex. AF_INET(Internet), AF_UNIX(UNIX), AF_NDD (Network Device Drivers)*
  - Tipo
    - *Ex. Stream, Datagram, Raw*
  - Protocolo
    - *Exemplos:*

AF_INET: TCP, UDP
AF_NDD: ATM

Tutorial para uso de raw sockets:
https://www.opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/

Domínios Utilizados (Família de Endereços)

O espaço no qual o endereço é especificado é chamado de domínio

Domínios básicos:

- INTERNET: `AF_INET` - os endereços consistem em endereço de rede (endereço IP) e número da porta, o que permite a comunicação entre processos de sistemas diferentes
- Unix: `AF_UNIX` - os processos comunicam-se referenciando um *pathname*, dentro do espaço de nomes do sistema de arquivos

***OBS**.: Com a criação do protocolo IPv6, há também a criação do domínio `AF_INET6`, capaz de representar endereçamento compatível ao IPv6.*

***Outros domínios:** `AF_ISO` (redes baseadas no padrão de protocolos ISO) e `AF_XNS` (Xerox Network Systems)*

Domínios Utilizados (Família de Endereços)

```
156      * Address families.
157      */
158     #define AF_UNSPEC       0               /* unspecified */
159     #define AF_LOCAL        1               /* local to host (pipes, portals) */
160     #define AF_UNIX         AF_LOCAL        /* backward compatibility */
161     #define AF_INET         2               /* internetwork: UDP, TCP, etc. */
162     #define AF_IMPLINK      3               /* arpanet imp addresses */
163     #define AF_PUP          4               /* pup protocols: e.g. BSP */
164     #define AF_CHAOS        5               /* mit CHAOS protocols */
165     #define AF_NS           6               /* XEROX NS protocols */
166     #define AF_ISO          7               /* ISO protocols */
167     #define AF_OSI          AF_ISO
168     #define AF_ECMA         8               /* european computer manufactu
169     #define AF_DATAKIT      9               /* datakit protocols */
170     #define AF_CCITT        10              /* CCITT protocols, X.25 etc *
171     #define AF_SNA          11              /* IBM SNA */
172     #define AF_DECnet       12              /* DECnet */
173     #define AF_DLI          13              /* DEC Direct data link interf
174     #define AF_LAT          14              /* LAT */
175     #define AF_HYLINK       15              /* NSC Hyperchannel */
176     #define AF_APPLETALK    16              /* Apple Talk */
177     #define AF_ROUTE        17              /* Internal Routing Protocol *
178     #define AF_LINK         18              /* Link layer interface */
179     #if defined(__EXT_BSD)
180     #define pseudo_AF_XTP   19              /* eXpress Transfer Protocol (
181     #endif
```

Domínios (famílias de endereços padrão estão definidos em sys/socket.h

AF = *Address Family*

```
182     #define AF_COIP         20              /* connection-oriented IP, aka ST II */
183     #define AF_CNT          21              /* Computer Network Technology */
184     #if defined(__EXT_BSD)
185     #define pseudo_AF_RTIP  22              /* Help Identify RTIP packets */
186     #endif
187     #define AF_IPX          23              /* Novell Internet Protocol */
188     #define AF_INET6        24              /* IP version 6 */
189     #if defined(__EXT_BSD)
190     #define pseudo_AF_PIP   25              /* Help Identify PIP packets */
191     #endif
192     #define AF_ISDN         26              /* Integrated Services Digital Network*/
193     #define AF_E164         AF_ISDN         /* CCITT E.164 recommendation */
194     #define AF_NATM         27              /* native ATM access */
195     #define AF_ARP          28              /* (rev.) addr. res. prot. (RFC 826) */
196     #if defined(__EXT_BSD)
197     #define pseudo_AF_KEY   29              /* Internal key management protocol  */
198     #define pseudo_AF_HDRCMPLT 30           /* Used by BPF to not rewrite hdrs
199                                                in interface output routine */
200     #endif
201     #define AF_BLUETOOTH    31
202     #define AF_IEEE80211    32              /* IEEE80211 */
```

Outros domínios: *AF_ISO (redes base*
(Xerox Network Systems)

Domínio Internet – AF_INET

Faz uso da implementação Unix dos protocolos TCP/UDP/IP

Consiste de:

- Endereço de rede da máquina (endereço IP – ex.: 192.168.1.99)
- Identificação do número da porta

Porta: *Inteiro de 16 bits (definido pelo usuário) – portas 1 a 1023 são exclusivas do sistema para propósitos específicos.*

Ex.: Telnet (porta 23), FTP (porta 21), SMTP (25), HTTP (80), DNS *lookup* (53)

Permite a comunicação entre processos distribuídos

Tipos de *Sockets*

Stream Sockets (Fluxo de Dados) – Suporta comunicação bidirecional, com fluxo de dados sequencial e confiável

Garante que o dado enviado não será perdido, duplicado, ou reordenado sem a indicação de que um erro tenha acontecido

Mensagens grandes são fragmentadas, transmitidas e reconstruídas no destino

Stream Sockets

- Especificadas pelo tipo: `SOCK_STREAM`
- Podem ser associadas aos domínios `AF_INET` e `AF_UNIX`
  - No domínio `AF_UNIX`, funciona da mesma forma que um *pipe*.
  - No domínio `AF_INET`, são implementadas por conexões TCP/IP:
    - IP faz o roteamento (encaminhamento) de pacotes através da rede, de um computador para outro
    - TCP implementa o sequenciamento de pacotes, controle de fluxo e retransmissão

Datagram Sockets

Datagram Sockets (Datagramas) – Suporta comunicação bidirecional, sem estabelecer ou manter conexões

Não há qualquer garantia de que dado enviado não será perdido, duplicado, ou reordenado

Há limite no tamanho das mensagens a serem enviadas

Especificadas pelo tipo: `SOCK_DGRAM`

Comunicação via TCP

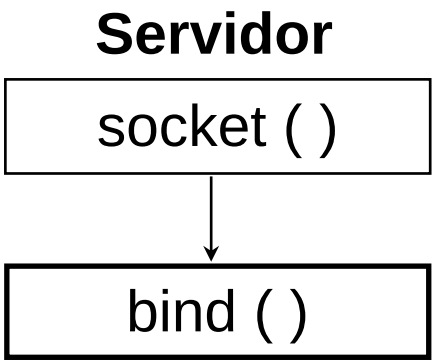
Servidor

socket ( )

Cria um *socket* com
- Família (ou domínio): UNIX, Internet, XNS
- Tipo: stream, datagrama, puro
- Protocolo (por conseq.): TCP, UDP

```
sockfd = (int) socket (int family, int type, int protocol)
```

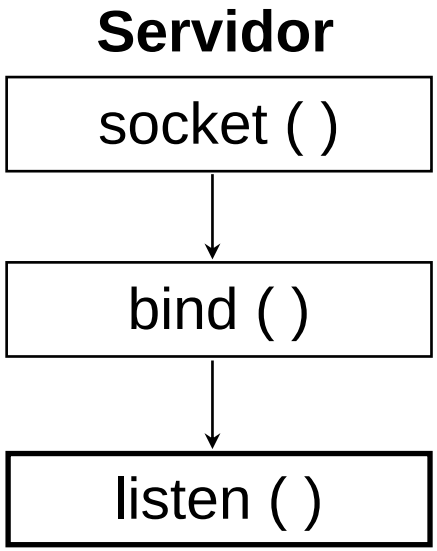
Comunicação via TCP



Atribui ao *socket*
- Endereço Internet (pode ser “any”)
- Porta de comunicação

```
ret = (int) bind (int sockfd, struct sockaddr *myaddr, int addrlen)
```

Comunicação via TCP

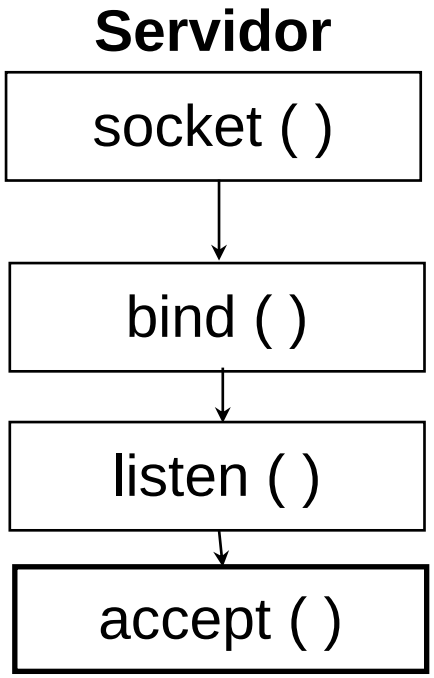


Declara
- Que está pronto para receber conexões
- Até quantas devem ser enfileiradas

Tamanho do backlog usado no Linux pode ser configurado no arquivo:
/proc/sys/net/ipv4/tcp_max_syn_backlog
O valor padrão é 256

```
ret = (int) listen (int sockfd, int backlog)
```

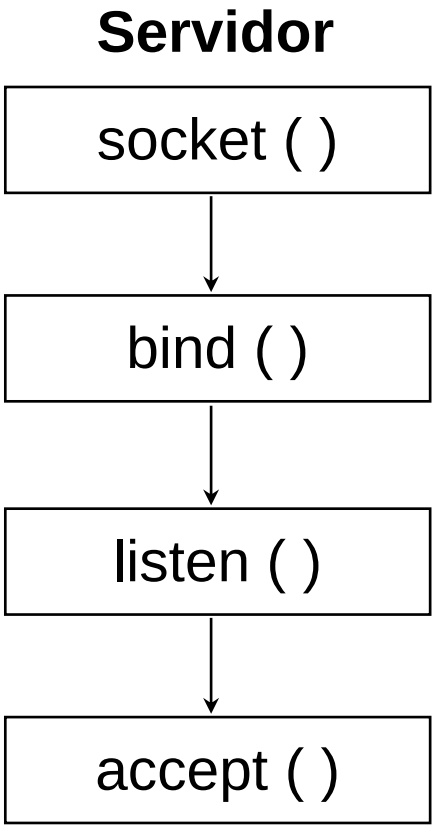
Comunicação via TCP



- Bloqueia até que haja pedido de conexão
- Quando houver algum, aceita

```
newsock = (int)accept(int sockfd, struct sockaddr *peer, int *addrlen)
```

Comunicação via TCP



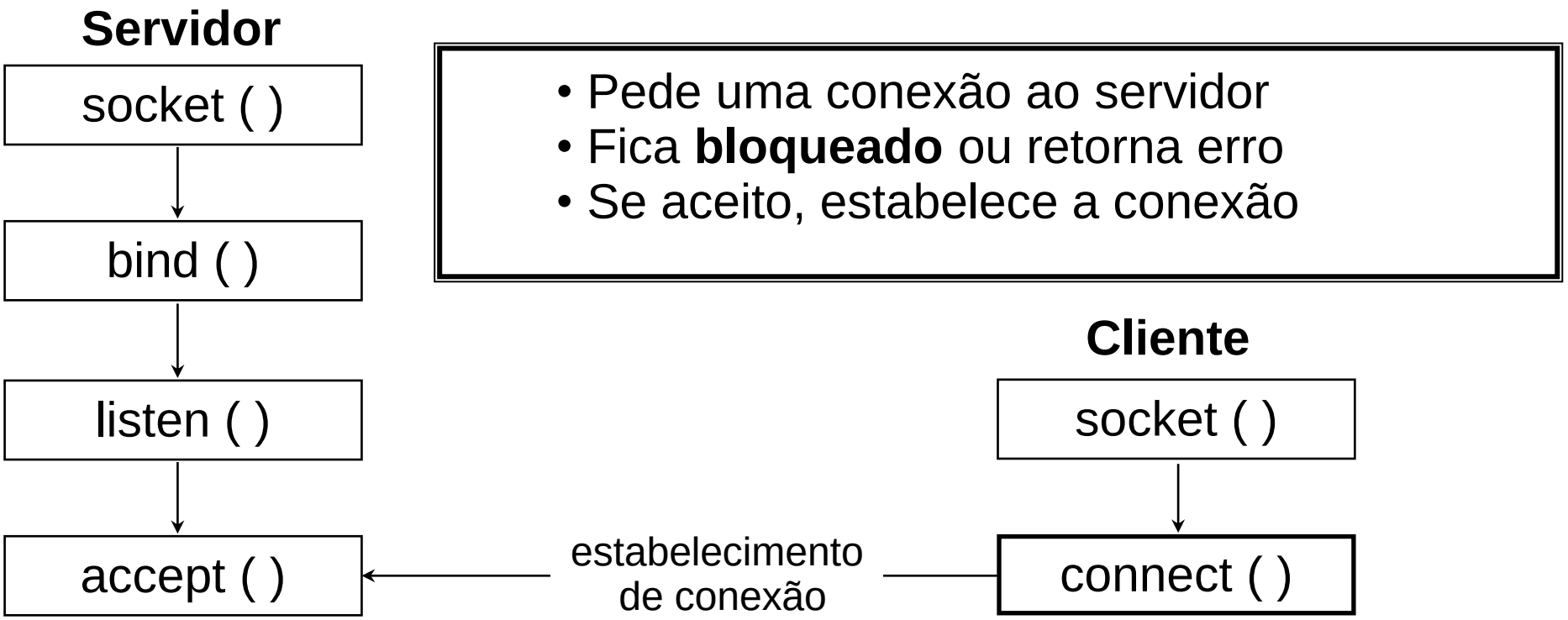
- Cria um *socket* idêntico ao do servidor (mesma família e tipo)

Cliente

socket ( )

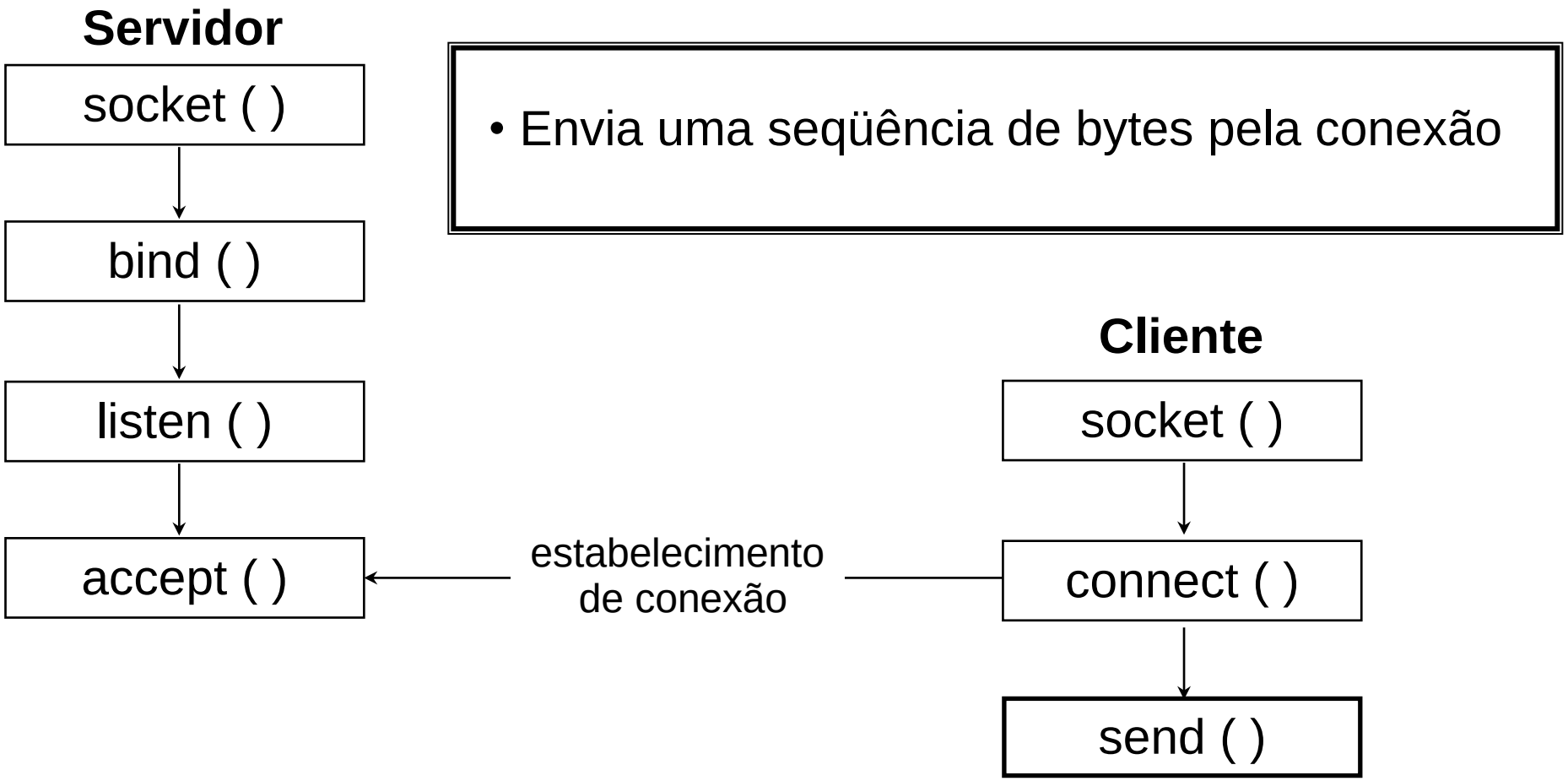
```
sockfd = (int) socket (int family, int type, int protocol)
```

Comunicação via TCP



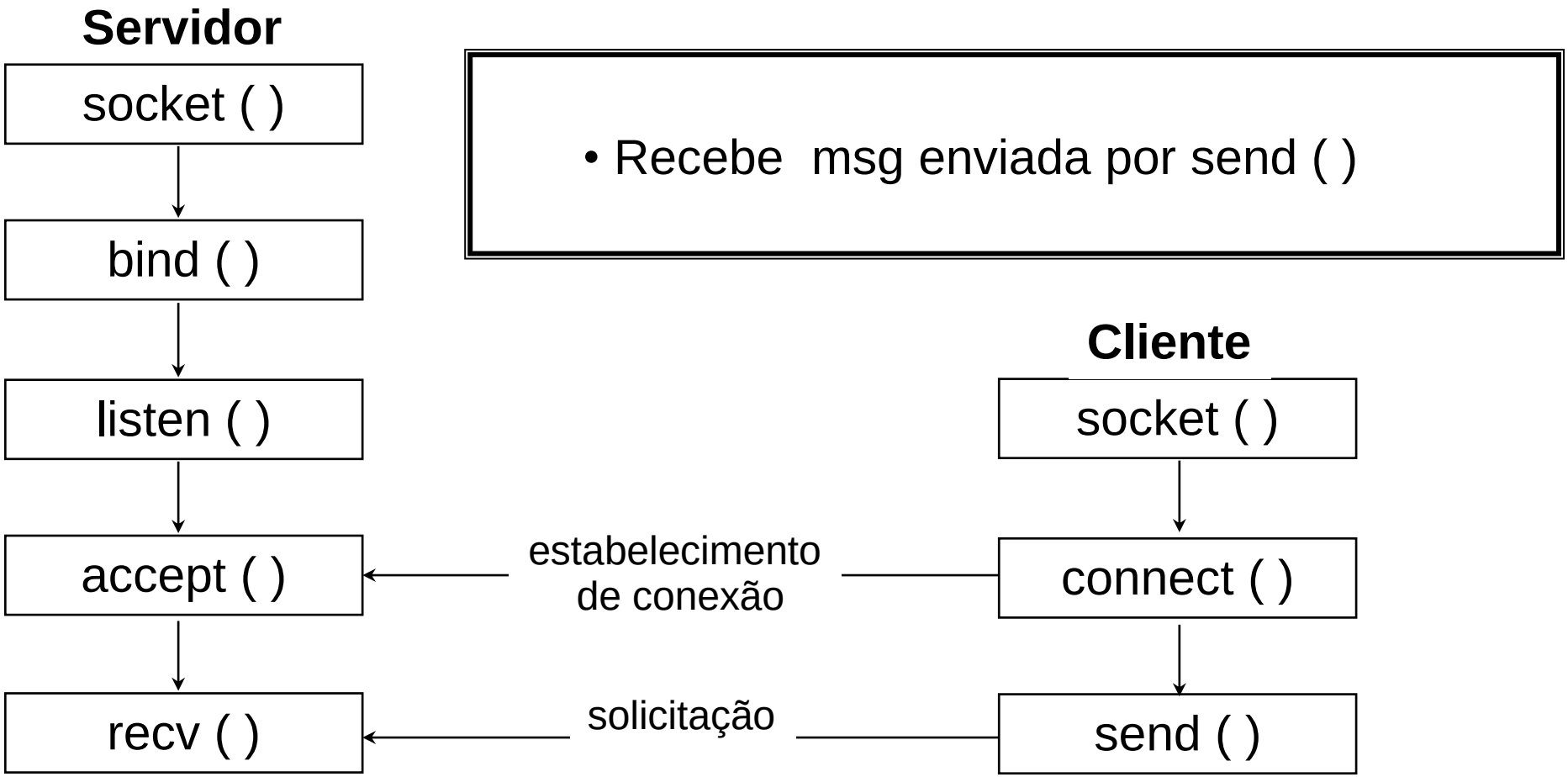

```
ret = (int)connect(int sockfd, struct sockaddr *servaddr, int addrlen)
```

Comunicação via TCP




```
nbytes = (int)send(int sockfd, char *buf, int nbytes, int flags)
```

Comunicação via TCP




```
nbytes =(int)recv(int sockfd, char *buf, int nbytes, int flags)
```

Comunicação via TCP

Alternativamente às primitivas send e receive, pode-se usar write e read (comuns também para leitura e escrita em arquivos:

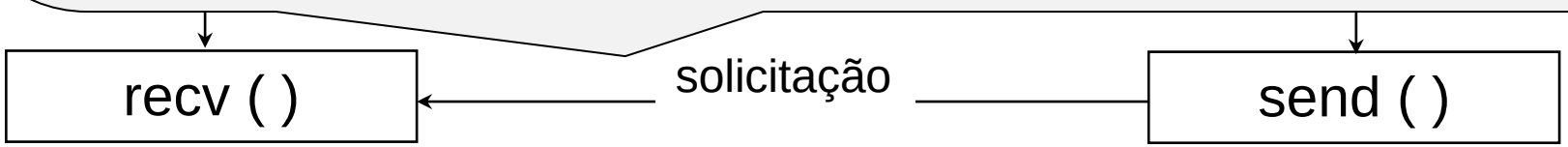
```
int write(int s, char *buf, int len)
```

s: descritor do socket
buf: endereço do *buffer* contendo o dado a ser escrito
len: tamanho em número de bytes a partir do endereço do *buffer*

```
int read(int s, char *buf, int len)
```

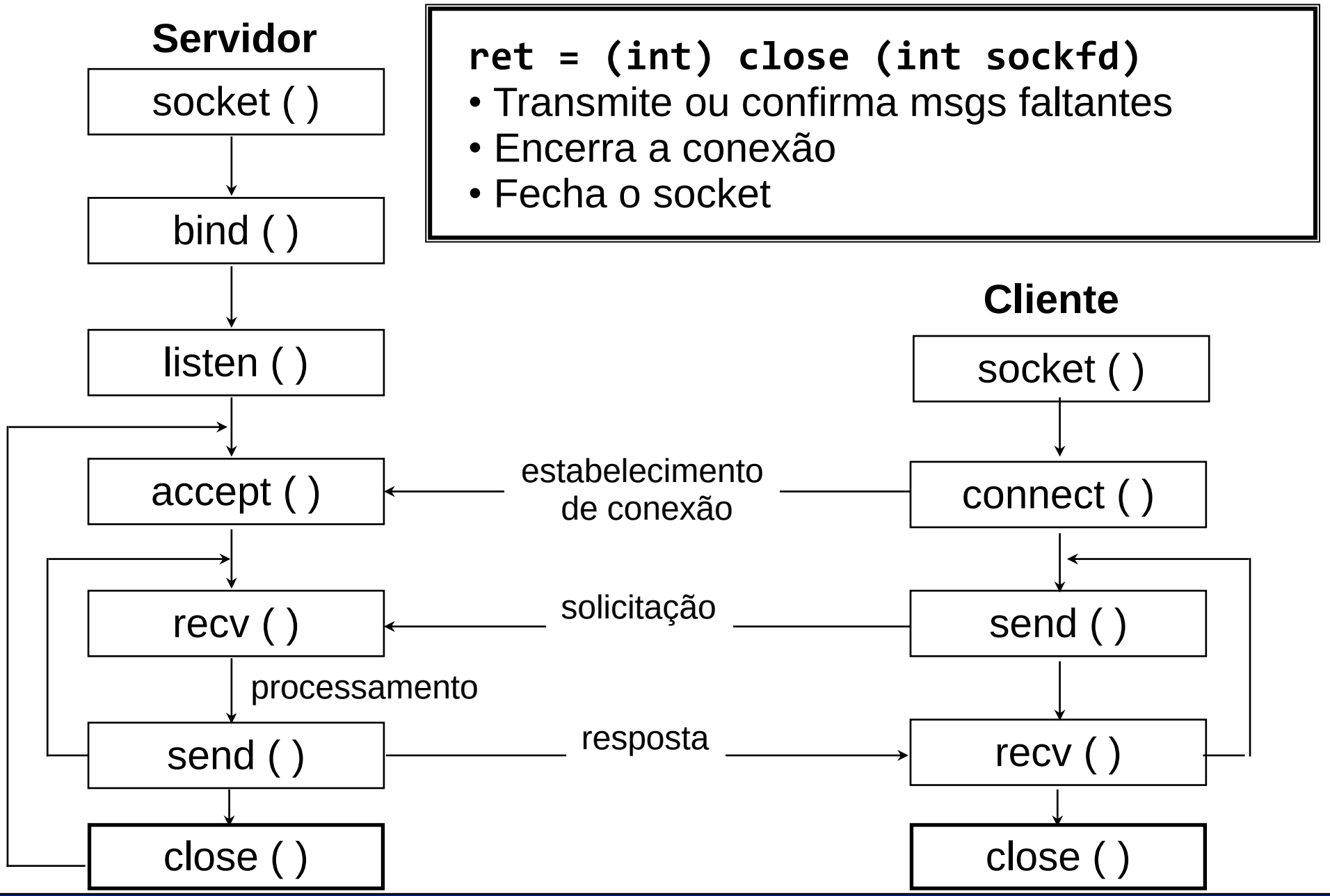
s: descritor do socket
buf: endereço do *buffer* que receberá o dado
len: tamanho em número de bytes a partir do endereço do *buffer*

O valor de retorno indica o número de bytes escritos ou lidos na operação




```
nbytes =(int)recv(int sockfd, char *buf, int nbytes, int flags)
```

Comunicação via TCP



Comunicação via UDP

- Cliente e servidor criam seus sockets
- Família = Internet, tipo = datagrama

Servidor

socket ( )

Cliente

socket ( )

Comunicação via UDP

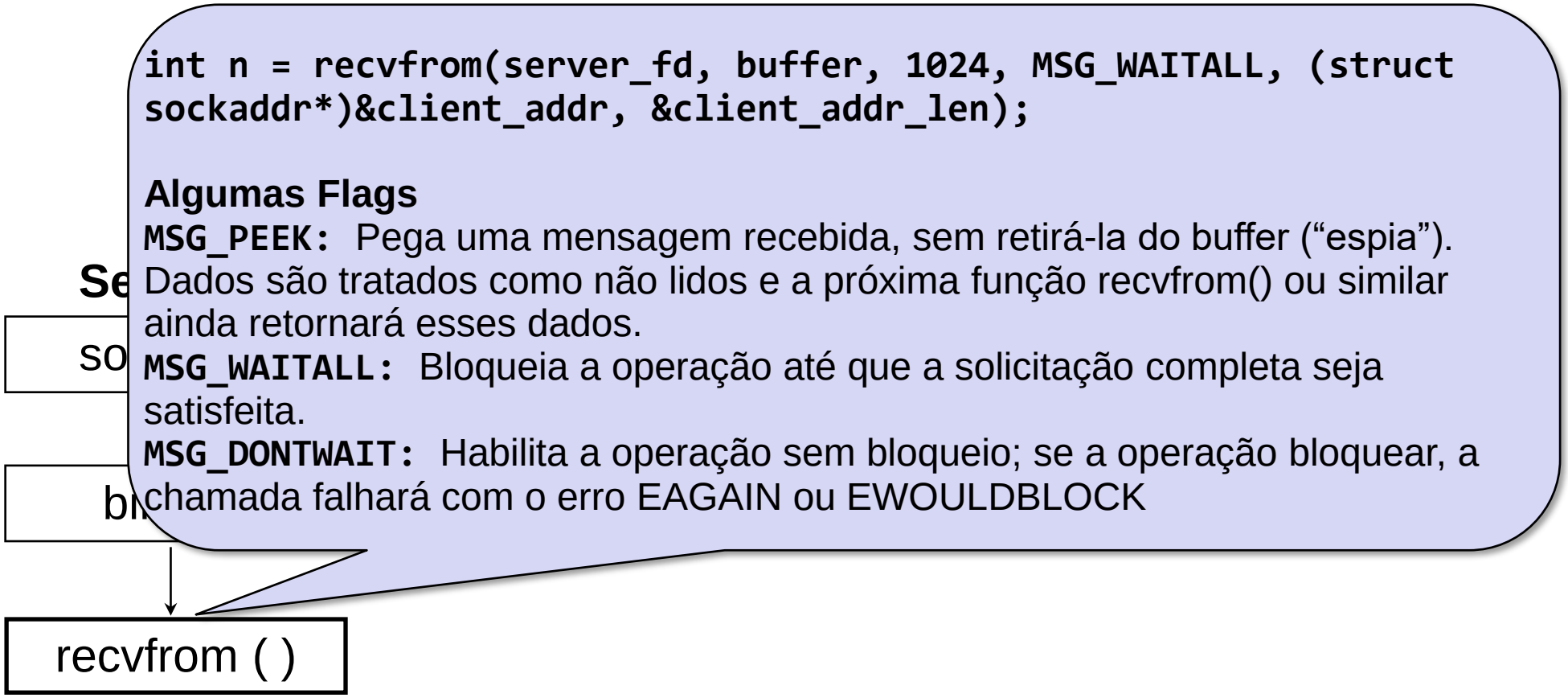
- Cliente e servidor definem endereços

Comunicação via UDP

- Recebe pacote enviado do endereço informado
- Se não houver nada, bloqueia


```
nbytes = (int)recvfrom(int sockfd, char *buf, int nbytes, int flags,
                       struct sockaddr *from, int *addrlen)
```

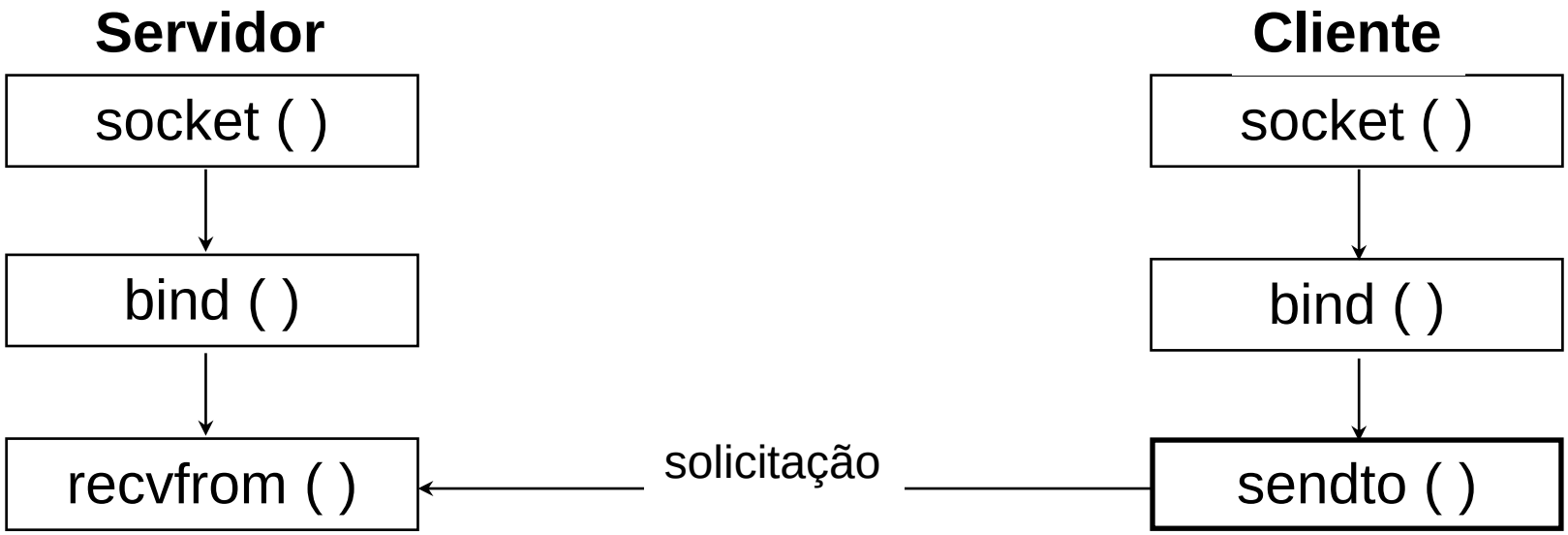
Comunicação via UDP




```
nbytes = (int)recvfrom(int sockfd, char *buf, int nbytes, int flags,
                       struct sockaddr *from, int *addrlen)
```

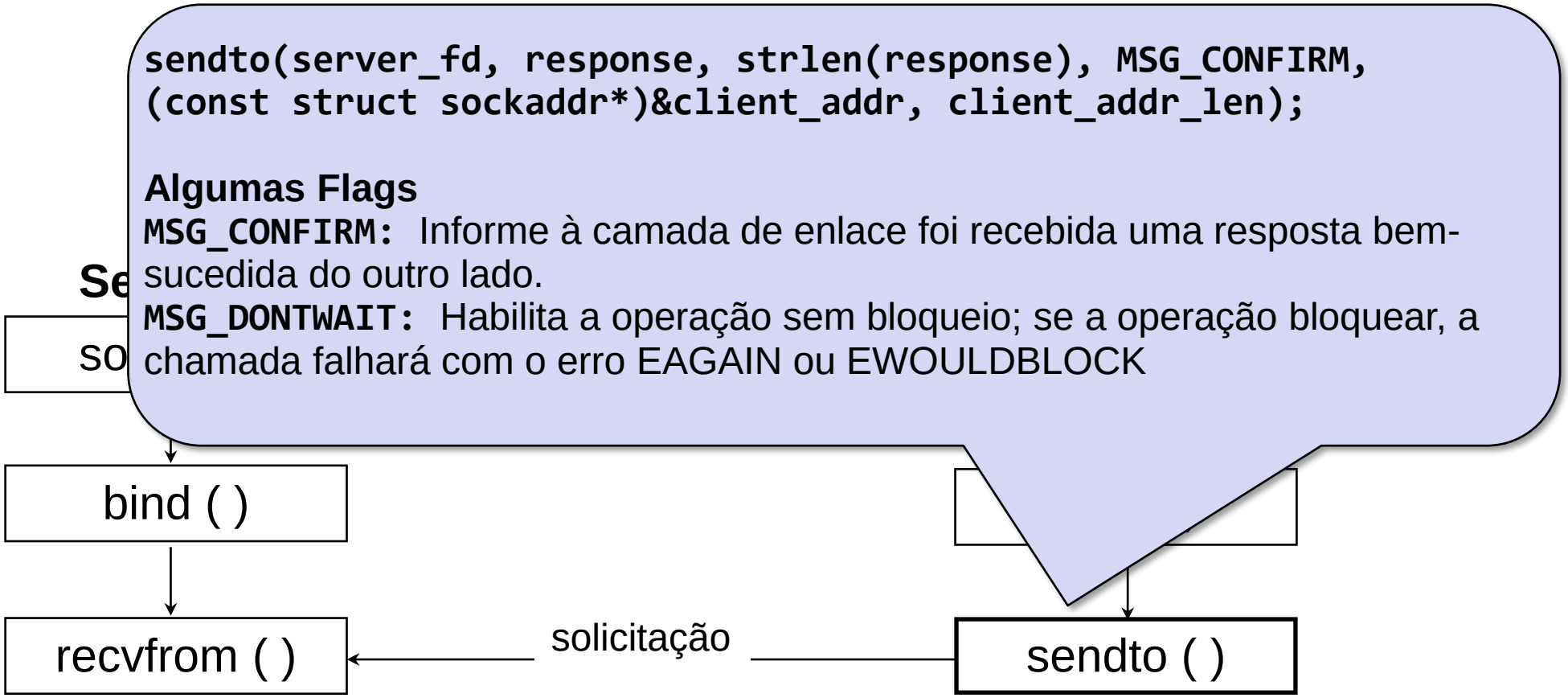
Comunicação via UDP

- Envia pacote para o endereço informado




```
nbytes = (int)sendto(int sockfd, char *buf, size_t len, int flags,
                     struct sockaddr *to, socklen_t tolen)
```

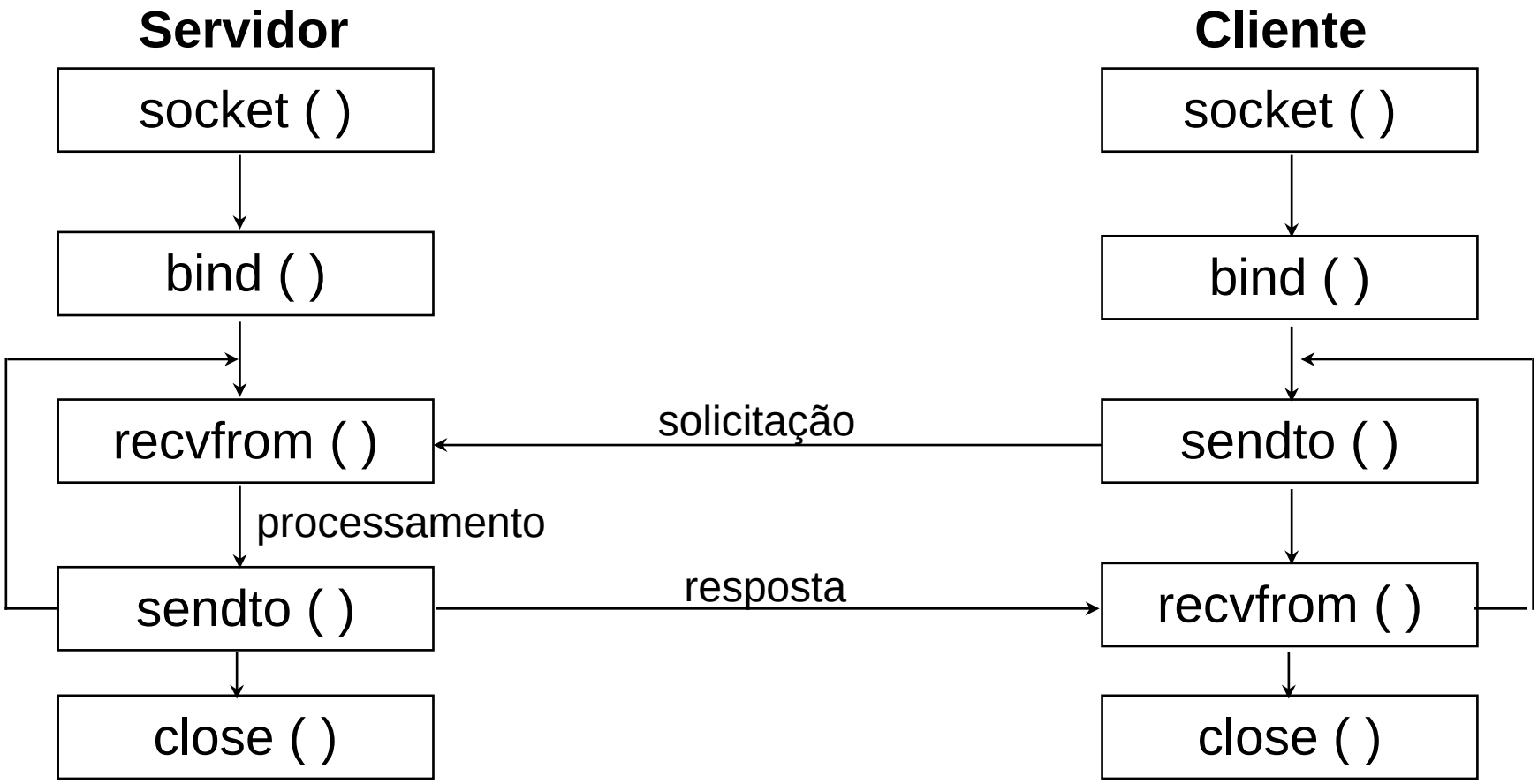
Comunicação via UDP




```
nbytes = (int)sendto(int sockfd, char *buf, size_t len, int flags,
                     struct sockaddr *to, socklen_t tolen)
```

Comunicação via UDP

- Diagrama completo



Estruturas Relacionadas à *Sockets* em C

Endereço Socket para Unix (AF_UNIX)

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

struct sockaddr_un {
sa_family_t sun_family; /* AF_UNIX */
char sun_path[]; /* pathname */
};
```

Endereço Socket para Rede (AF_INET)

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

struct sockaddr_in {
short int sin_family; /* AF_INET */
unsigned short int sin_port; /* Port number */
struct in_addr sin_addr; /* Internet address */
};
```

Endereço IP para Socket

```
struct in_addr {
unsigned long int s_addr;
};
```

Estruturas Relacionadas à *Sockets* em C

Criação de Socket

A chamada ao sistema cria um descritor para o *socket*.

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

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

Nomeação de Socket

Para estar disponível, o *socket* precisa receber uma referência (nome). Associa o endereço ao *socket*, o tamanho do endereço varia em AF_UNIX.

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

int bind(int socket, const struct sockaddr *address, size_t address_len);
```

Fila de Socket

Cria-se uma fila de *socket* para aceitar novas conexões que chegam. Linux limita o número de conexões pendentes. Backlog pode ser usado para "segurar" conexões adicionais pendentes.

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

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

Fechar um Socket

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

int close(int socket);
```

Códigos de Erros Relacionados à *Sockets* em C

AF_UNIX

Erro	Descrição
EACCESS	Não pode criar arquivo devido às permissões
ENOTDIR, ENAMETOOLONG	Indica uma forma inválida para o nome de arquivo

AF_INET

Erro	Descrição
EBADF	Descritor de arquivo inválido
ENOTSOCK	Descritor de arquivo não referencia um socket
EINVAL	Descritor de arquivo referencia um socket já nomeado
EADDRNOTAVAIL	Endereço indisponível
EADDRINUSE	Endereço já possui um socket utilizando seus limites

Estruturas Relacionadas à *Sockets* em C

Aceitar Conexões

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

int accept(int socket, struct sockaddr *address, size_t *address_len);
```

- Esta chamada ao sistema deve ser feita após o *socket* ser criado, nomeado e ter uma fila de conexão alocada para ele;

- O cliente será a primeira conexão pendente na fila;

- O parâmetro **`address_len`** especifica o tamanho da estrutura do cliente. Se o endereço do cliente é maior do que especificado, ele será truncado

- Se não há conexões pendentes na fila, **`accept`** ficará bloqueado até clientes solicitarem uma conexão

- Para tornar **`accept`** não-bloqueante:

```
int flags = fcntl(socket, F_GETFL, 0);
fcntl(socket, F_SETFL, O_NONBLOCK|flags);
```

Estruturas Relacionadas à *Sockets* em C

Solicitar Requisições

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

int connect(int socket,const struct sockaddr *address,size_t address_len);
```

- Se a conexão não puder ser estabelecida imediatamente, **`connect`** bloqueará por um período não especificado. Se um limite de espera (timeout) for atingido, a conexão será abortada e connect irá falhar

- Códigos de Erro para **`connect`**:

Erro	Descrição
EBADF	Descritor de arquivo inválido foi passado ao *socket*
EALREADY	Já há uma conexão em progresso para o *socket*
ETIMEDOUT	*Timeout* para a tentativa de conectar
ECONNREFUSED	Conexão recusada pelo servidor

- Para tornar **`connect`** não-bloqueante:

```
int flags = fcntl(socket, F_GETFL, 0);
fcntl(socket, F_SETFL, O_NONBLOCK|flags);
```

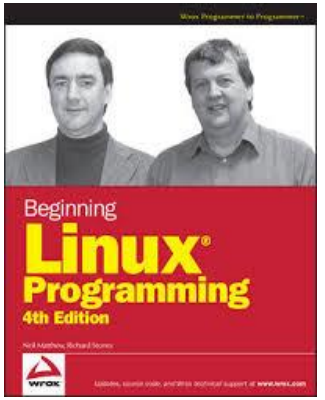
Exemplo de Código – Cliente Local (AF_UNIX)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
     int sockfd;
     int len;
     struct sockaddr_un address;
     int result;
     char ch = ‘A’;

     sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
     address.sun_family = AF_UNIX;
     strcpy(address.sun_path, “server_socket”);
     len = sizeof(address);
     result = connect(sockfd, (struct sockaddr *)&address, len);
     if(result == -1) {
          perror(“oops: client1”);
          exit(1);
     }
     write(sockfd, &ch, 1);
     read(sockfd, &ch, 1);
     printf(“char from server = %c\n”, ch);
     close(sockfd);
     exit(0);
}
```

Exemplos adaptados de:



Exemplo de Código – Servidor Local (AF_UNIX)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
     int server_sockfd, client_sockfd;
     int server_len, client_len;
     struct sockaddr_un server_address;
     struct sockaddr_un client_address;
     char ch;

     server_sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
     server_address.sun_family = AF_UNIX;
     strcpy(server_address.sun_path, “server_socket”);
     server_len = sizeof(server_address);
     bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
     listen(server_sockfd, 5);
     while(1) {
          printf(“server waiting\n”);
          client_len = sizeof(client_address);
          client_sockfd = accept(server_sockfd,(struct sockaddr *)&client_address, &client_len);
          read(client_sockfd, &ch, 1);
          ch++;
          write(client_sockfd, &ch, 1);
          close(client_sockfd);
     }
}
```

Exemplo de Código – Execução

Executando o servidor

```
$ ./server1 &
[1] 1094
$ server waiting

$ ls -lF server_socket
srwxr-xr-x 1 odo None 0 2011-03-21 15:21 server_socket=

$ ps lx
F UID PID PPID PRI NI VSZ RSS WCHAN STAT TTY TIME COMMAND
0 1000 23385 10689 17 0 1424 312 361800 S pts/1 0:00 ./server1
```

Executando o cliente

```
$ ./client1
server waiting
char from server = B
```

Executando múltiplos clientes

```
$ ./client1 & ./client1 & ./client1 &
[2] 23412
[3] 23413
[4] 23414
server waiting
char from server = B
server waiting
char from server = B
server waiting
char from server = B
server waiting
[2] Done client1
[3]- Done client1
[4]+ Done client1
```

Exemplo de Código – Cliente *Stream* (AF_INET)

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

int main() {
     int sockfd;
     int len;
     struct sockaddr_in address;
     int result;
     char ch = ‘A’;

     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     address.sin_family = AF_INET;
     address.sin_addr.s_addr = inet_addr("127.0.0.1");
     address.sin_port = htons(9734);
     len = sizeof(address);
     result = connect(sockfd, (struct sockaddr *)&address, len);
     if(result == -1) {
          perror(“oops: client1”);
          exit(1);
     }
     write(sockfd, &ch, 1);
     read(sockfd, &ch, 1);
     printf(“char from server = %c\n”, ch);
     close(sockfd);
     exit(0);
}
```

Exemplo de Código – Servidor *Stream* (AF_INET)

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

int main() {
     int server_sockfd, client_sockfd;
     int server_len, client_len;
     struct sockaddr_in server_address;
     struct sockaddr_in client_address;
     char ch;

     server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
     server_address.sin_family = AF_INET;
     server_address.sin_addr.s_addr = htonl(INADDR_ANY);
     server_address.sin_port = htons(9734);
     server_len = sizeof(server_address);
     bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
     listen(server_sockfd, 5);
     while(1) {
          printf("server waiting\n");
          client_len = sizeof(client_address);
          client_sockfd = accept(server_sockfd,(struct sockaddr *)&client_address, &client_len);
          read(client_sockfd, &ch, 1);
          ch++;
          write(client_sockfd, &ch, 1);
          close(client_sockfd);
     }
}
```

Exemplo de Código – Cliente UDP (AF_INET)

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

#define PORT 6000
int main() {
    int sockfd;
    struct sockaddr_in s_addr;
    char *message = "Hello, here is a UDP client";
    char buffer[1024];

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    s_addr.sin_family = AF_INET;
    s_addr.sin_port = htons(PORT);
    s_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    sendto(sockfd, message, strlen(message), MSG_CONFIRM, (struct sockaddr*)&s_addr, sizeof(s_addr));
    printf("UDP message sent to server\n");
    socklen_t len = sizeof(s_addr);
    int n = recvfrom(sockfd, buffer, 1024, MSG_WAITALL, (struct sockaddr*)&s_addr, &len);
    buffer[n] = '\0';
    printf("Message received from server: %s\n", buffer);

    close(sockfd);
    return 0;
}
```

Exemplo de Código – Servidor UDP (AF_INET)

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

#define PORT 6000
int main() {
    int server_fd;    char buffer[1024];
    struct sockaddr_in s_addr, c_addr;    socklen_t c_addr_len;
    char *response = "Hello from UDP server";

    if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("Socket creation failed");    exit(EXIT_FAILURE);
    }
    s_addr.sin_family = AF_INET;
    s_addr.sin_addr.s_addr = INADDR_ANY;
    s_addr.sin_port = htons(PORT);
    if (bind(server_fd, (const struct sockaddr*)&s_addr, sizeof(s_addr)) < 0) {
        perror("Bind failed");        close(server_fd);        exit(EXIT_FAILURE);
    }
    printf("Server waiting..\n");
    c_addr_len = sizeof(c_addr);
    int n = recvfrom(server_fd, buffer, 1024, MSG_WAITALL, (struct sockaddr*)&c_addr, &c_addr_len);
    buffer[n] = '\0';
    printf("Message from client: %s\n", buffer);
    sendto(server_fd, response, strlen(response), MSG_CONFIRM, (const struct sockaddr*)&c_addr,
c_addr_len);
    printf("Response sent to client\n");
    close(server_fd);
    return 0;
}
```

Exercício 1 – *Echo*

Criar um programa cliente que conecte ao servidor executando no endereço e porta definidos abaixo.
O servidor implementa um serviço de echo, que basicamente retorna ao cliente a mensagem recebida.

O programa cliente deve imprimir na tela a mensagem de retorno do servidor.

Informações Adicionais:

```
Tipo de conexão: TCP ou UDP
Endereço do Servidor: Definir em aula
Porta: 1100
```

Exercício 2 – Calculadora

Criar um servidor de operações aritméticas. Definir um protocolo em aula para as operações `soma`, `subtracao`, `multiplicacao` e `divisao`.

O servidor deve atender requisições de vários clientes e retornar o valor destas operações.

Informações Adicionais:

```
As operações devem conter apenas dois operadores
O operando será informado pelo como cabeçalho do protocolo
Endereço do Servidor: Definir em aula
Porta: 1101
```

Exercício 3 – Monitor UDP

Criar um programa cliente que repetidamente gera um número aleatório entre 1 e 100 e envia o número para o servidor usando datagramas.

O servidor aguarda datagramas com valores de medição coletados por clientes. Esses valores são ilustrados por números aleatórios entre 1 e 100.

O servidor, a cada 1s, imprime o número de medições coletadas no último segundo e o valor médio de todas as medições.

Exercício 4 – Servidor com *pool* de *threads*

Criar um servidor usando TCP. A thread `main` aguarda por conexões e a cada conexão estabelecida, associa o socket retornado no `accept` a uma *thread* de um *pool* de *threads* trabalhadoras.

O *pool* de *threads* tem tamanho N_THREADS (ex. 4 *threads*)

Utilize um semáforo para o controle de *threads* disponíveis no pool. A *thread* `main`, a cada socket repassado para uma *thread* do *pool*, decrementa o semáforo. Caso não haja *threas* desocupadas, a thread `main` bloqueia no semáforo, não mantendo novos clientes pendentes.

Cada *thread* trabalhadora, ao encerrar a conexão com o cliente que esteja atendendo, incrementa o semáforo, indicando que está disponível para atender novos clientes

Referências utilizadas

Beginning Linux Programming, Neil Matthew & Richad Stones. Wrox Press, 2004.