Computação Distribuída

Odorico Machado Mendizabal

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

Sockets

Primitivas de Comunicação – *Sockets*

- Proposto 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 (TCP ou UDP)

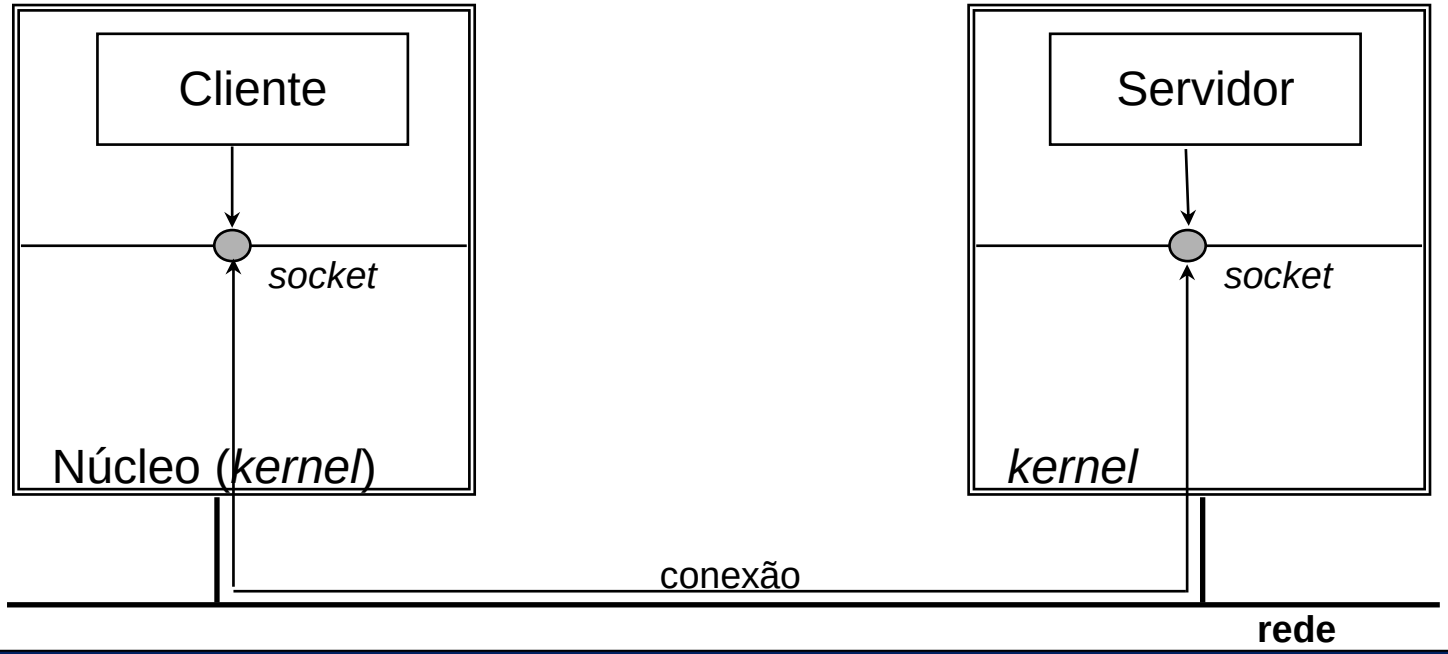
- Endereço de *socket*
  - Identificador de comunicação formado por um endereço de rede (ex. IP) e um identificador de porta local

Comunicação usando *Sockets*

• Operações de comunicação entre processos usam pares de *sockets*, 1 em cada lado da comunicação

• *Sockets* são criados por processos através de chamadas ao sistema

• Mensagens são enfileiradas no *socket* de envio até a transmissão e a chegada do ACK, e são enfileiradas no *socket* de destino até que o processo faça uma chamada de sistema para obter a mensagem.



Características

- *Sockets* permitem comunicação bi-direcional
  - Chamadas de envio e recebimento para mesmo *socket*

- Ligação do *socket* com o processo comunicante antes de realizar a comunicação

- Endereço do *socket* é público
  - `N` clientes enviam para o *socket* de um servidor

- *Sockets* devem ser fechados explicitamente

Atributos de um *Socket*

- *Sockets* são caracterizados por 3 atributos:
  - Domínio
  - Tipo
  - Protocolo

Domínios Utilizados

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)*

Domínio Internet – AF_INET

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

Consiste de:

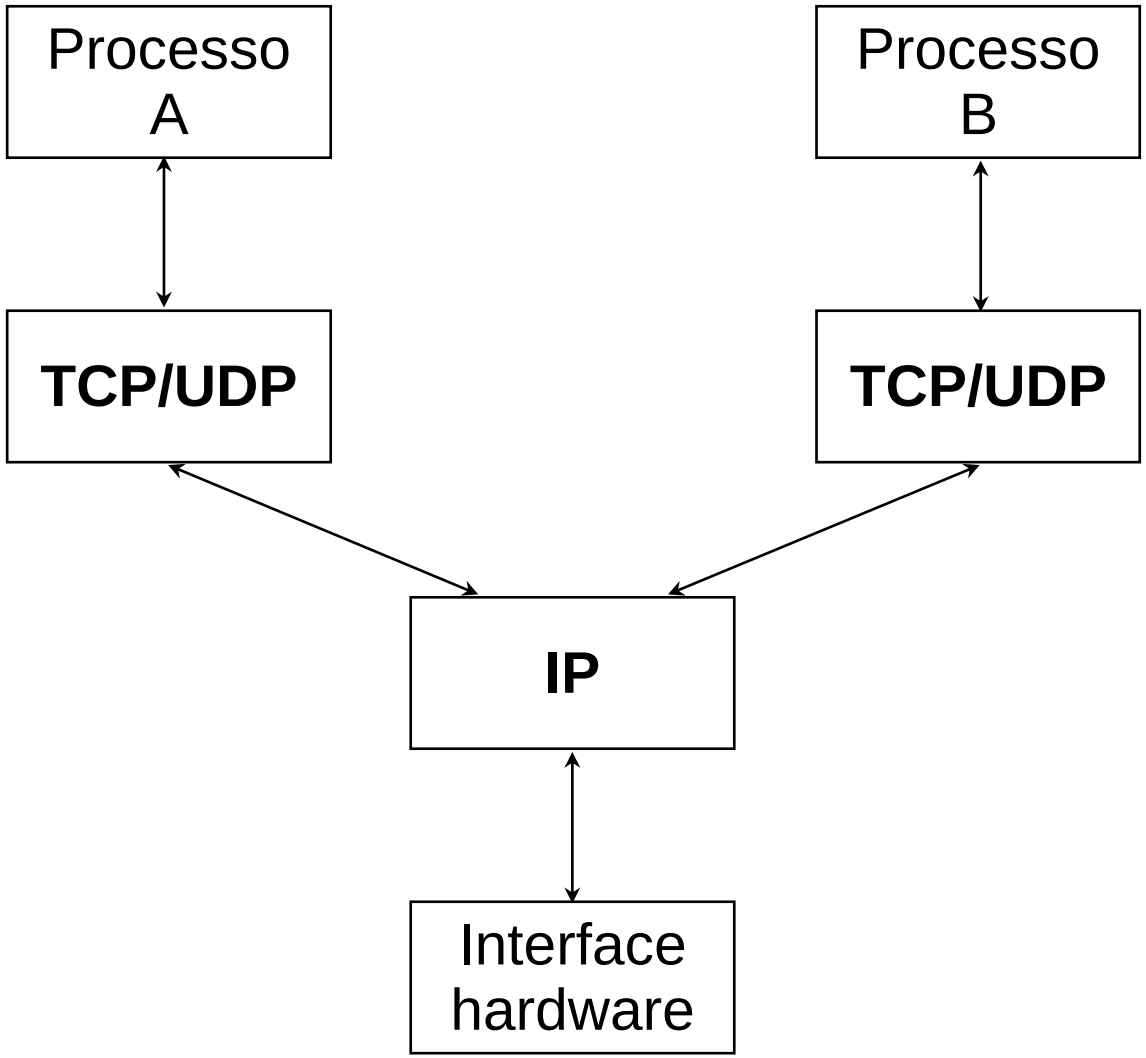
- endereço de rede da máquina (endereço IP – ex.: 192.168.1.99)
- identificação do número da porta

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)

Permite a comunicação entre processos distribuídos

Protocolos TCP/UDP/IP



Tipos de *Sockets*

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

Garantia que 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

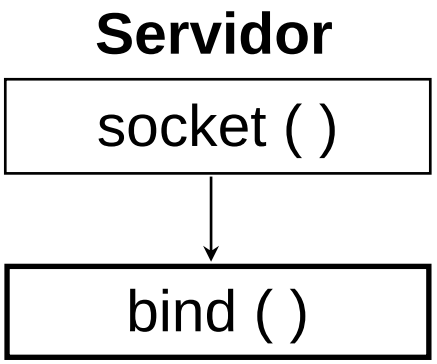
Servidor

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)
```

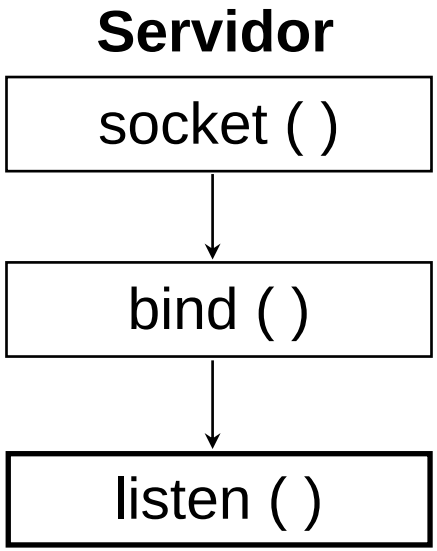
Comunicação via TCP



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

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

Comunicação via TCP

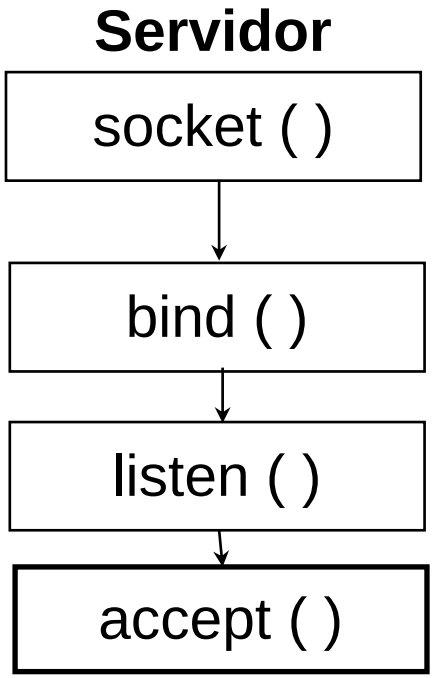


Declara
- Que está pronto para receber conexões
- Até quantas devem ser enfileiradas

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)
```

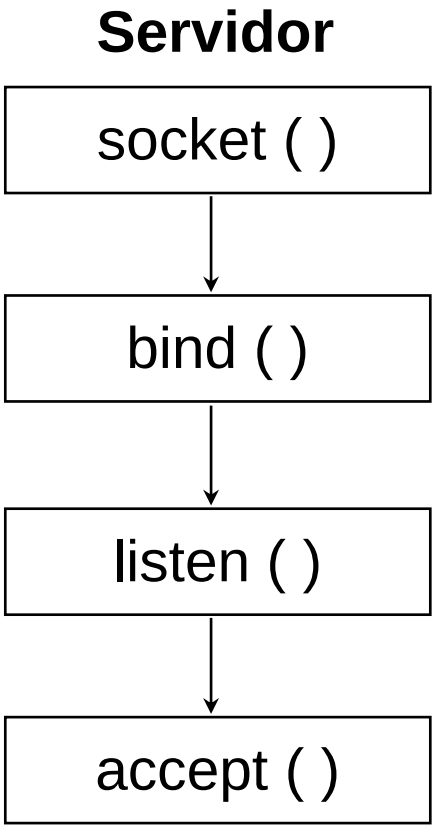
Comunicação via TCP



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

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

Comunicação via TCP



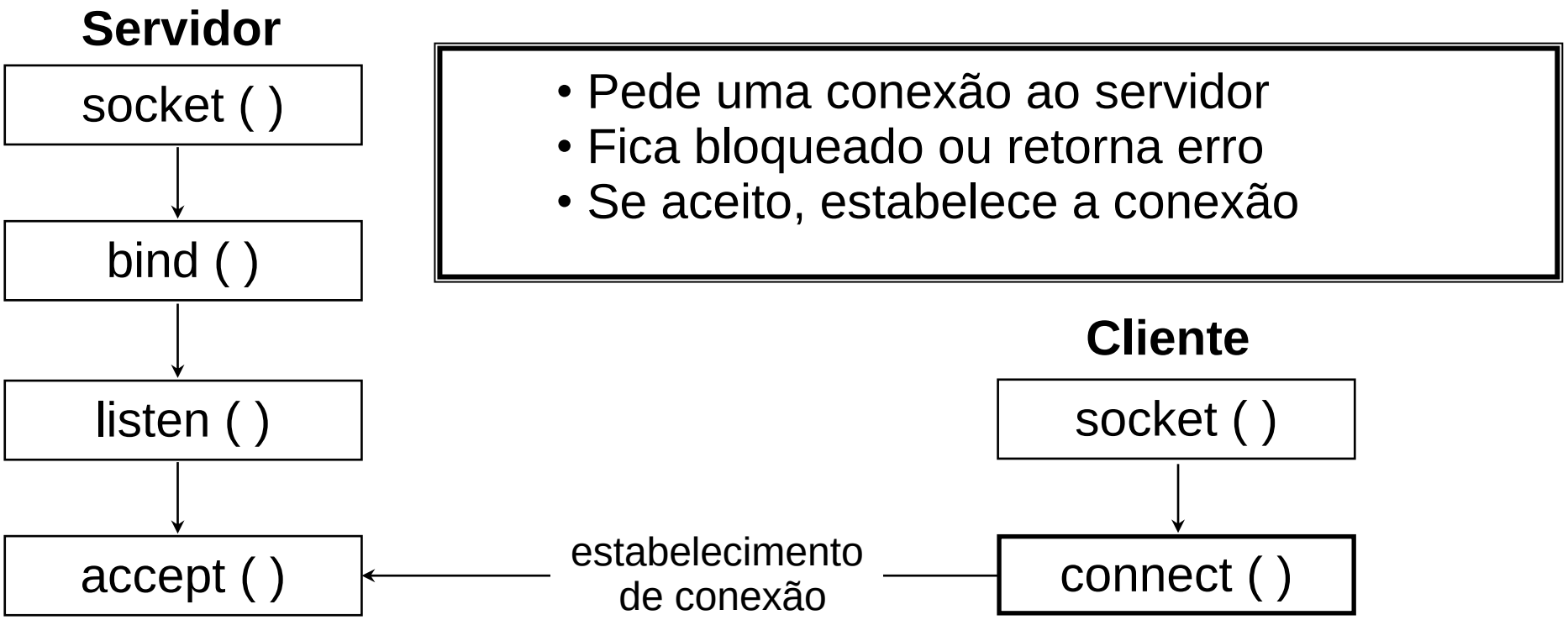
- Cria um *socket* idêntico ao do servidor (mesma família e tipo)

Cliente

socket ( )

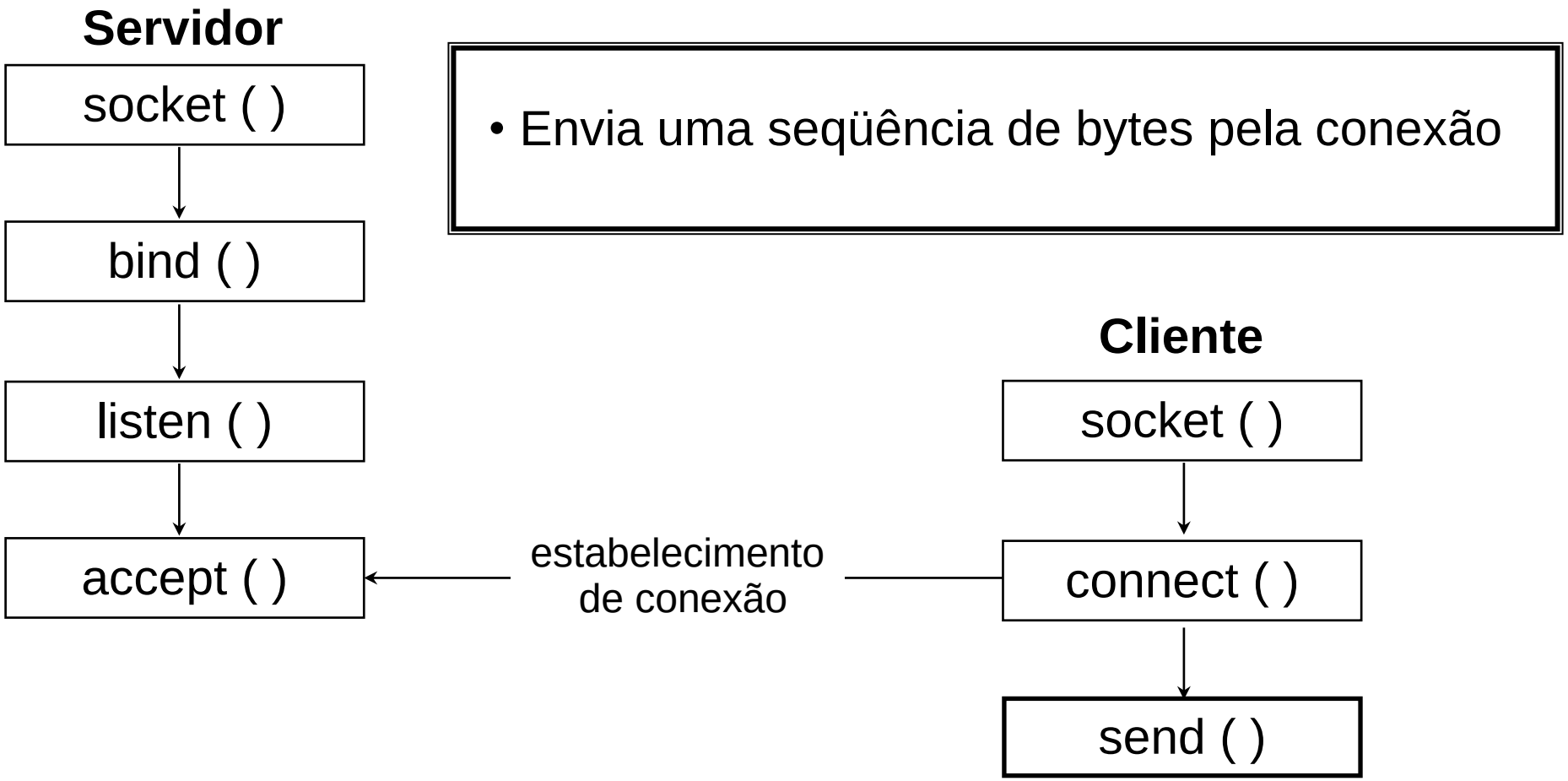
```
sockfd = (int) socket (int family, int type, int protocol)
```

Comunicação via TCP



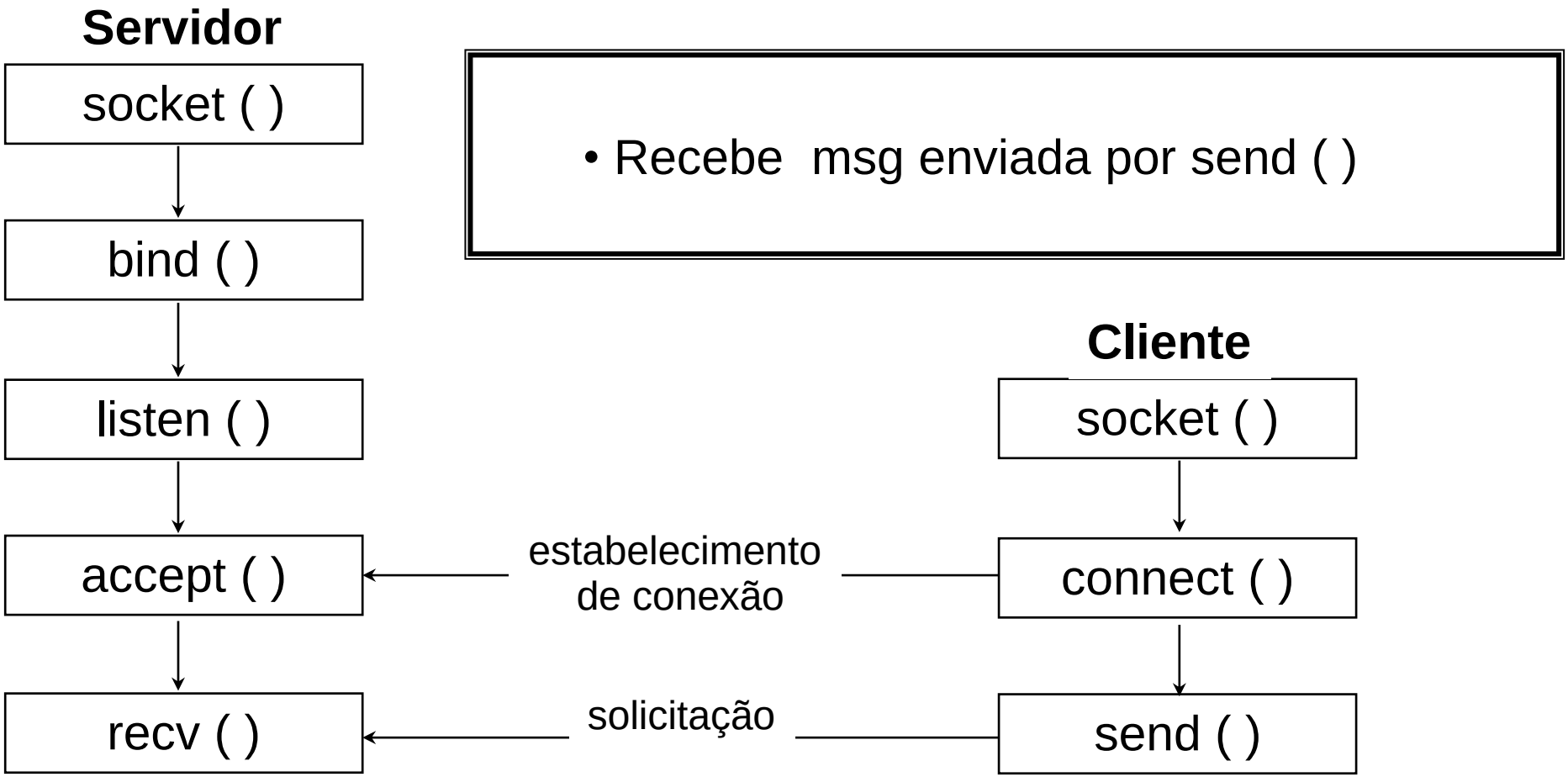

```
ret = (int)connect(int sockfd, struct sockaddr *servaddr, int addrlen)
```

Comunicação via TCP



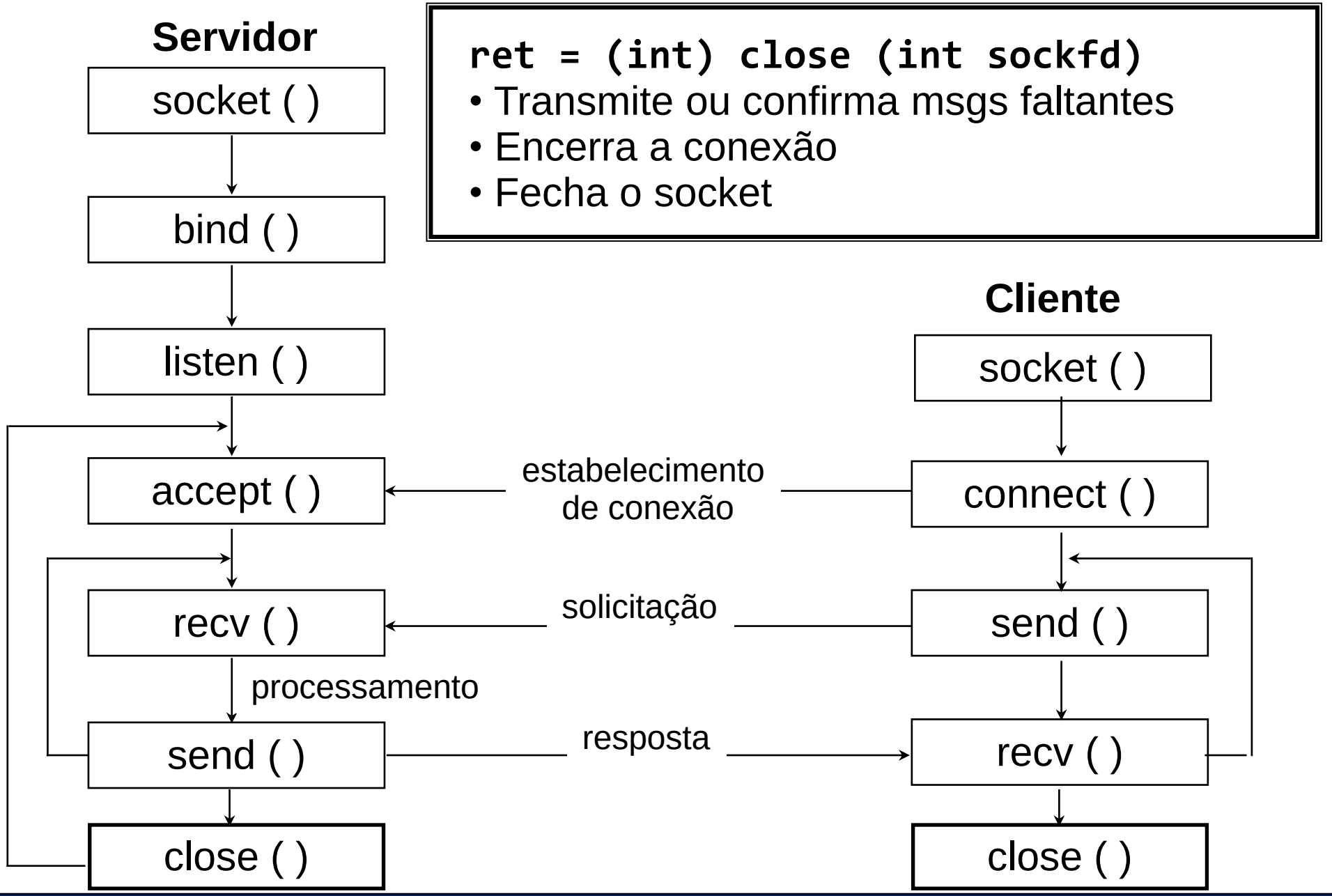

```
nbytes = (int)send(int sockfd, char *buf, int nbytes, int flags)
```

Comunicação via TCP

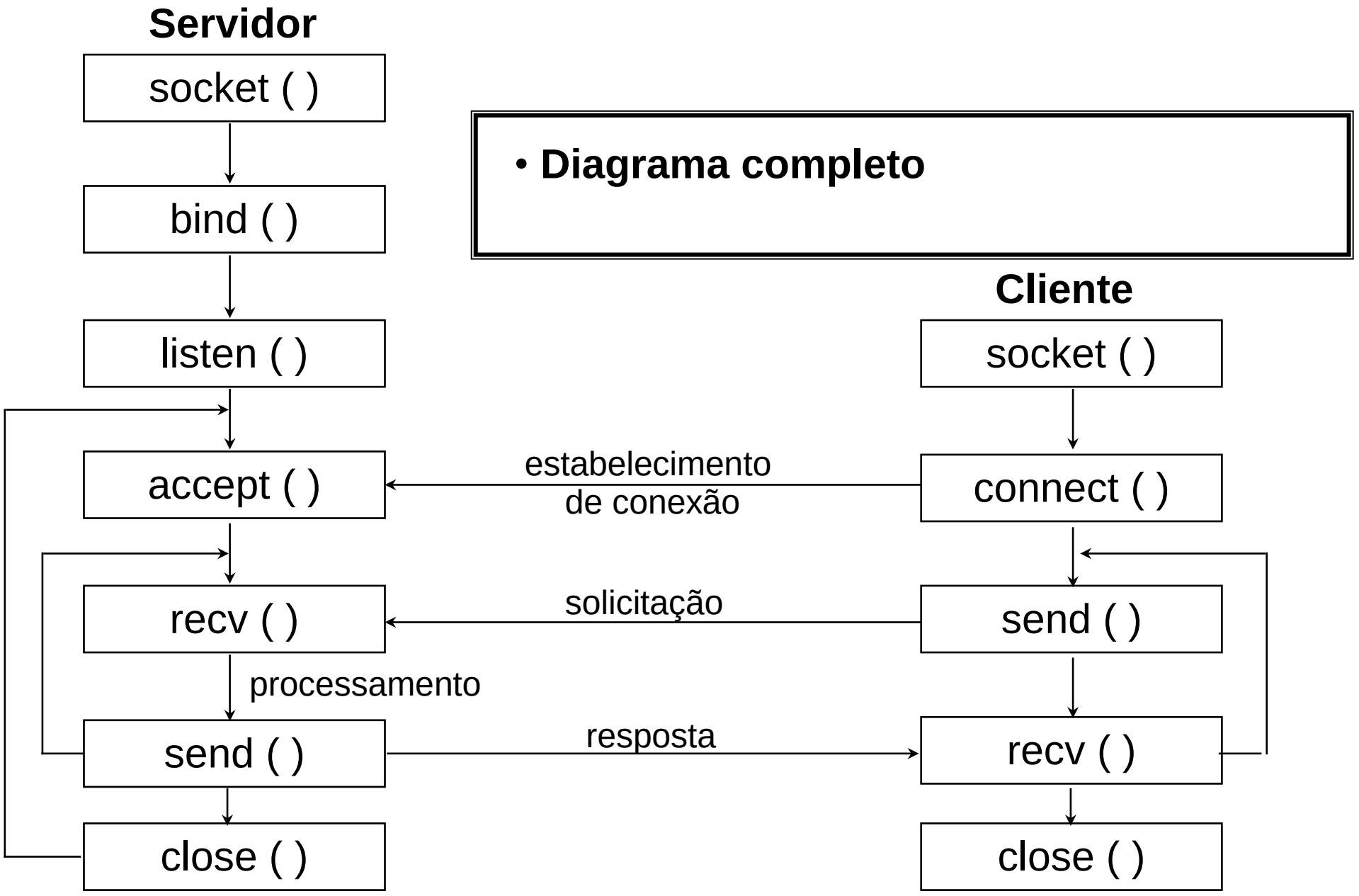



```
nbytes =(int)recv(int sockfd, char *buf, int nbytes, int flags)
```

Comunicação via TCP



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

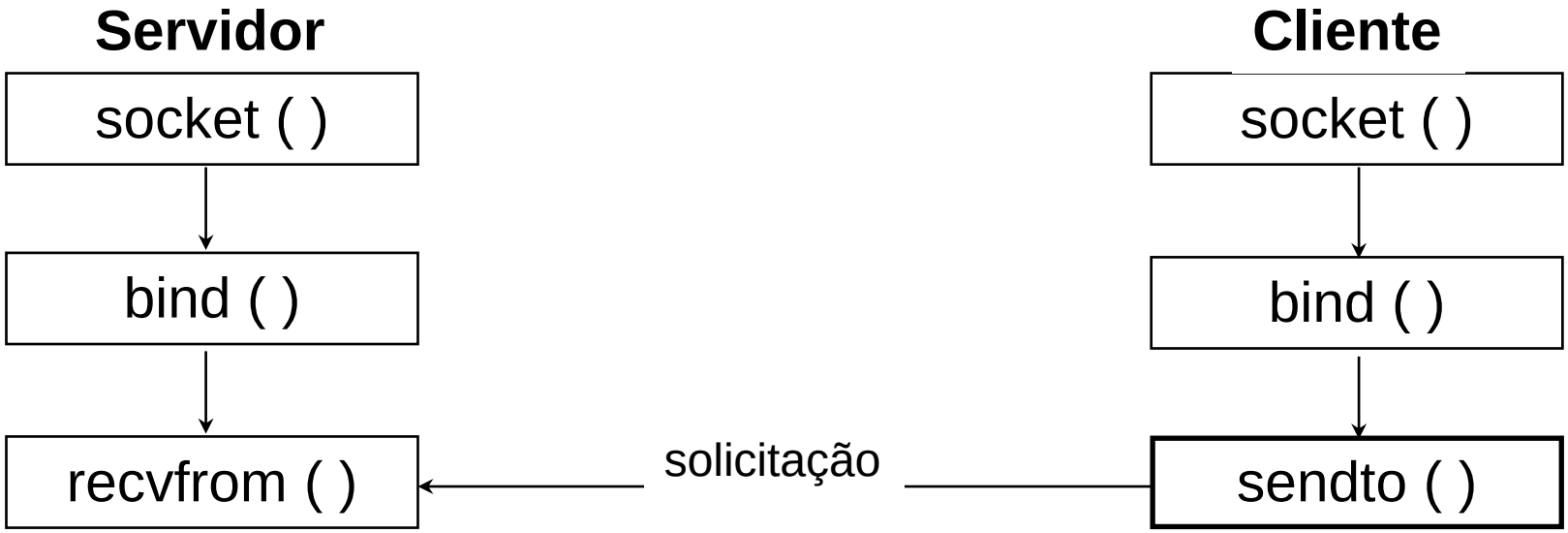
Comunicação via UDP

- 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

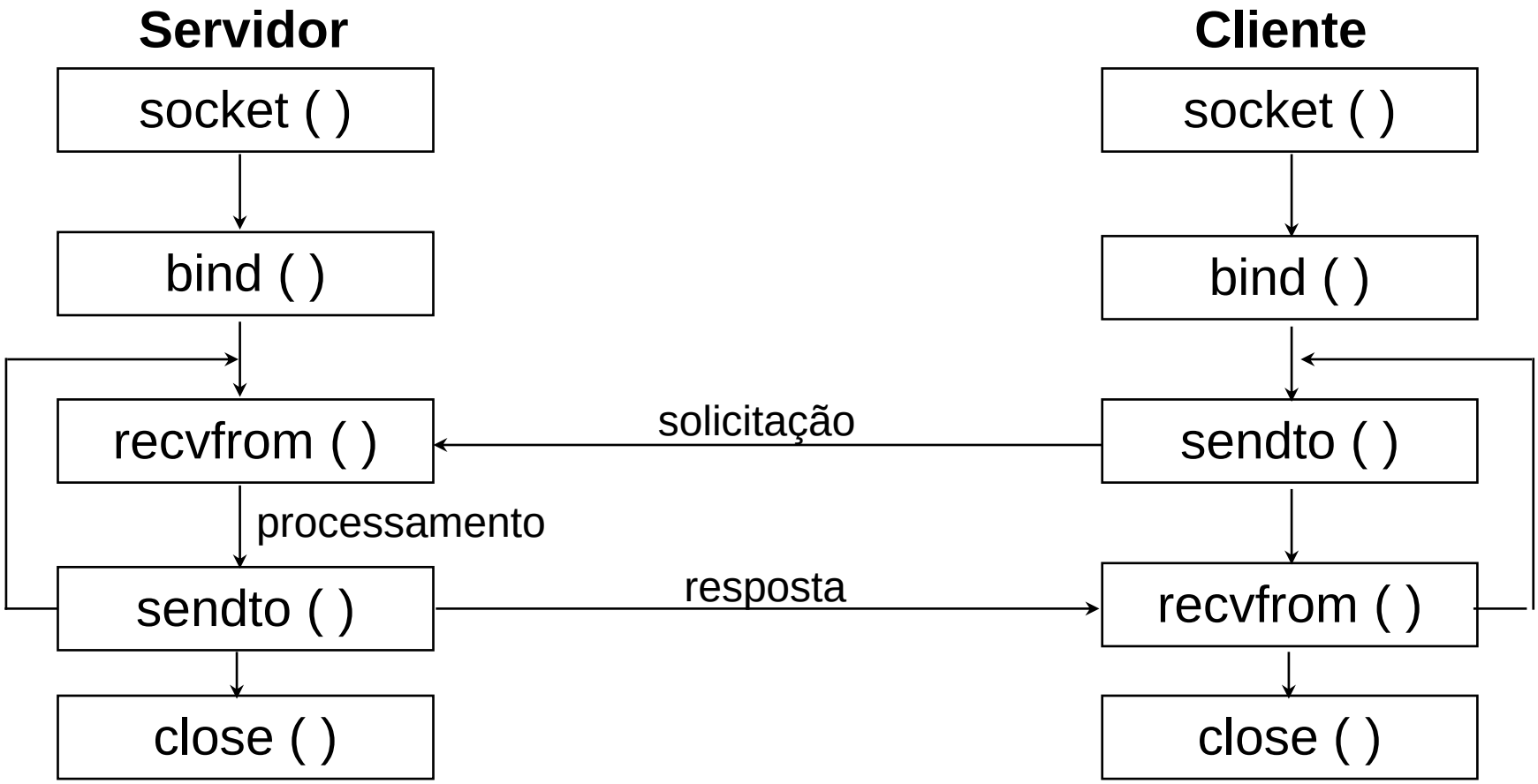
- Envia pacote para o endereço informado




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

Comunicação via UDP

- Diagrama completo



Sockets e Datagramas

Usa UDP - sem conexão

Endereço dos *sockets* enviados a cada mensagem

Não há ACK

recvfrom() bloqueia processo se *socket* vazio

```
s = socket(AF_INET, SOCK_DGRAM, 0)
        :
bind(s, ClientAddress)
        :
sendto(s, "message", ServerAddress)
```

Enviando a mensagem

```
s = socket(AF_INET, SOCK_DGRAM, 0)
        :
bind(s, ServerAddress)
        :
amount = recvfrom(s, buffer, from)
```

Recebendo a mensagem

Sockets e Streams

Usa TCP - conexão prévia entre pares de *sockets*

Fila vazia e fila cheia: bloqueio do receptor e emissor respectivamente

sNew() cria um novo *socket* para aceitar a conexão com um cliente e libera o *socket* original para novas conexões

```
s = socket(AF_INET, SOCK_STREAM, 0)
        :
connect(s, ServerAddress)
        :
write(s, "message", length)
```

Enviando a mensagem

```
s = socket(AF_INET, SOCK_STREAM, 0)
bind(s, ServerAddress)
listen(s,5)
sNew = accept(s, from)
n = read(sNew, buffer, amount)
```

Recebendo a mensagem

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 Relacionadas 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);
```

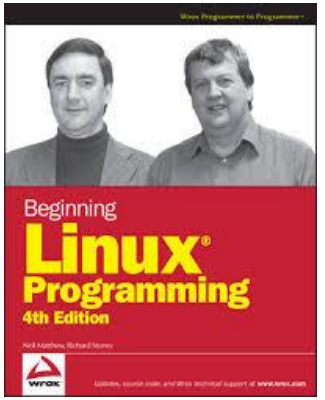
Exemplo de Código – Cliente Local (AF_UNIX)

```
#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 Rede (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 = 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 Rede (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 = 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);
     }
}
```

Referências utilizadas

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