II - UFRGS

Conceitos de Sockets

Valter Roesler

Universidade Federal do Rio Grande do Sul (UFRGS)

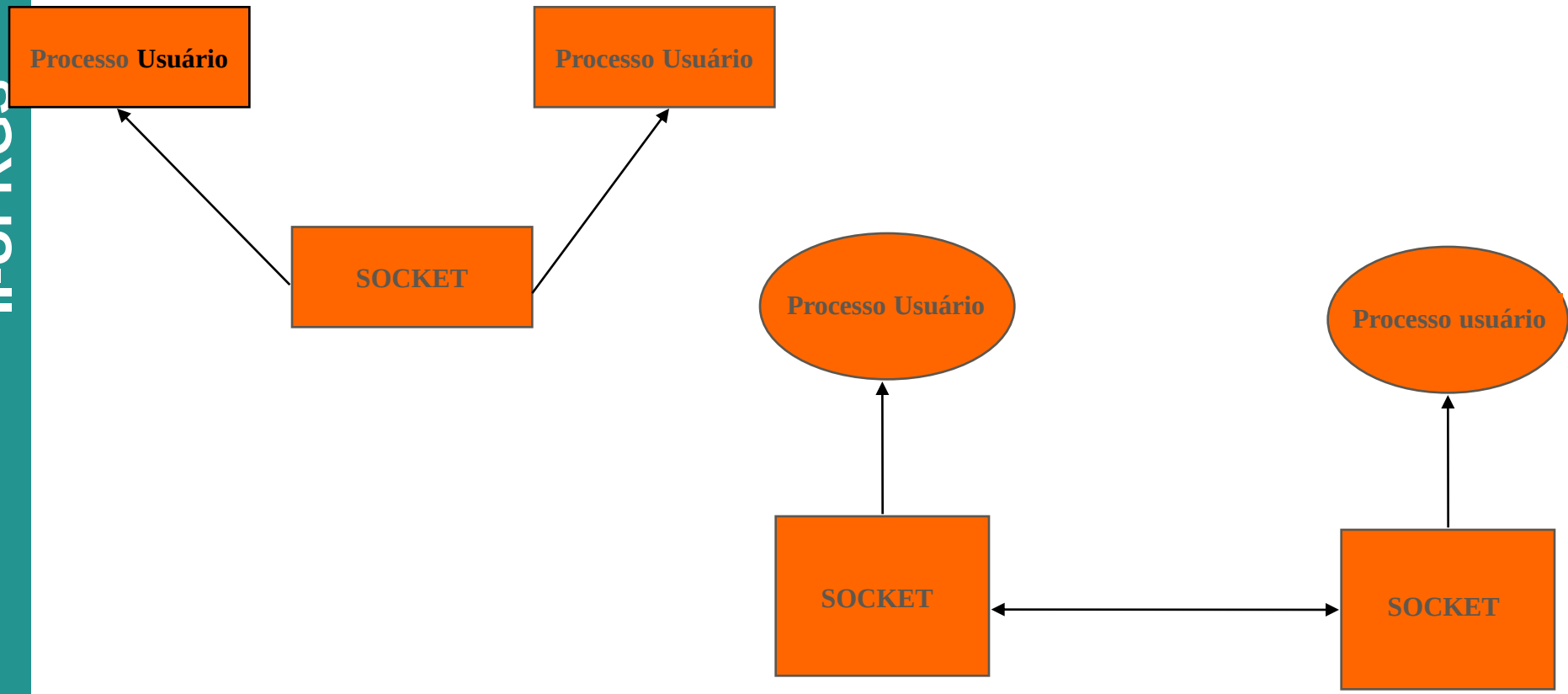
Instituto de Informática

Agenda

- Conceitos sobre sockets
- Sockets UDP
- Sockets TCP

Conceito de Sockets

- Uma forma de IPC (*Inter-process communication*).

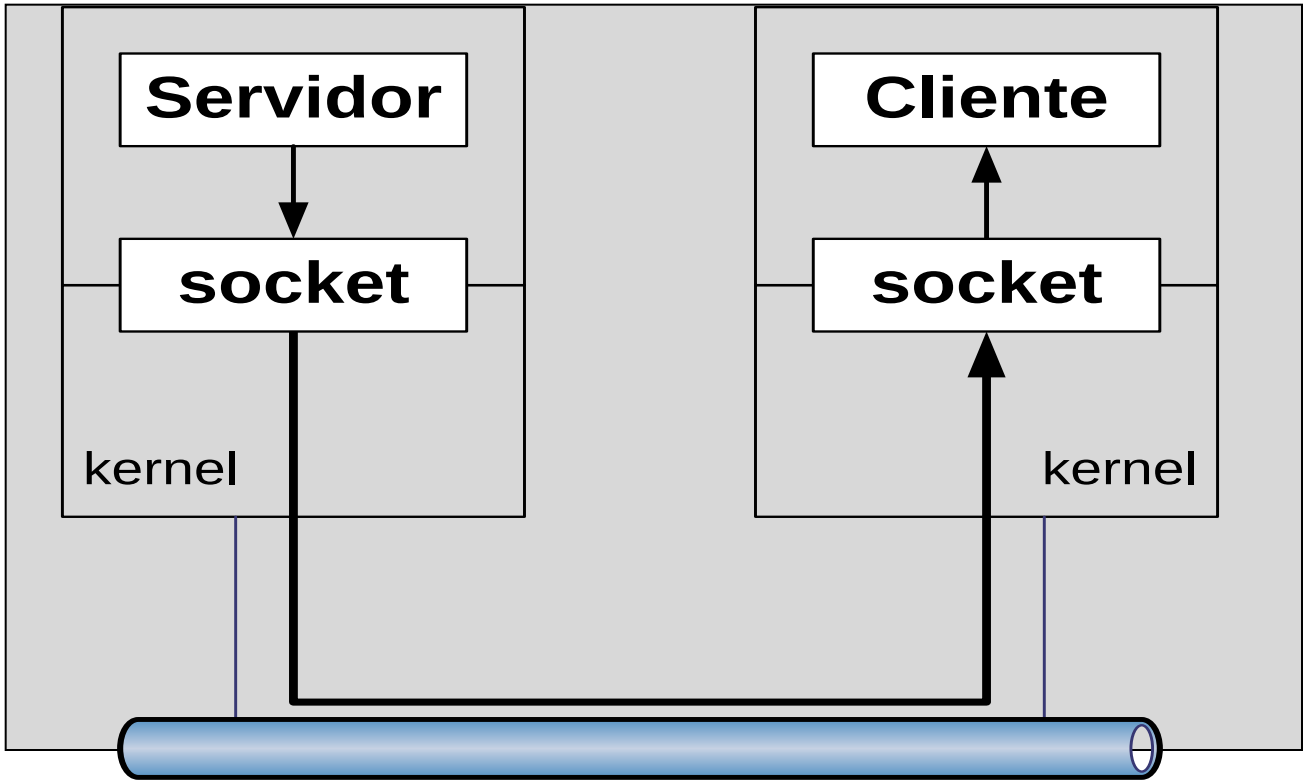


Base

- Onde ficam os sockets no modelo OSI?
- O que é fim-a-fim?
- O que é confiável e não confiável?
- O que é conexão / conexionless?
- O que é Datagrama?

Associação

- Os processos são ligados por uma associação formada por:
  - Protocolo (TCP/UDP)
  - Endereço e Porta
  - Endereço e Porta remota



Inicialização do Socket

Servidor

socket()

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

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

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

- int domain
  - AF_UNIX
  - AF_INET
  - AF_INET6
- int type
  - SOCK_STREAM
  - SOCK_DGRAM
  - SOCK_RAW
- int protocol
  - Depende do domínio. Normalmente se usa o valor ZERO

Domínios

- **AF_UNIX**: Usado no domínio local. Mesma máquina.
- **AF_INET**: Usado no domínio da Internet (IPv4). Podendo ser a mesma máquina ou em máquinas diferentes.
- **AF_INET6**: Mesmo domínio do AF_INET, porém usando o IPv6.

Tipos de Sockets (SOCK_DGRAM)

- Comunicação não confiável.
- Usa o protocolo UDP
- Envio das mensagens não garantidamente seqüenciais.
- Usado em transmissões *Multicast*, transmissão de vídeos, ...

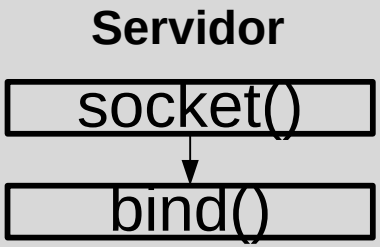
Tipos de Sockets (SOCK_STREAM)

- Fornece uma comunicação confiável.
- Usa o protocolo TCP.
- As mensagens são seqüenciadas.
- Uso adequado para transmissões de dados que requeiram confiabilidade. Exemplo:
  - Transferência de arquivos (FTP, HTTP);
  - Envio de e-mail (SMTP, POP3)
  - Login Remoto (SSH)

Tipos de Sockets (SOCK_RAW)

- Permite acesso do pacote diretamente à aplicação, eliminando a passagem pelo encapsulamento TCP/IP. Assim, a aplicação é responsável por eliminar e tratar os cabeçalhos.
- Pode ser usado para criar um sniffer de redes (entretanto, normalmente os analisadores de rede utilizam a API do pcap / winpcap).

Inicialização no servidor
(quem aguarda a mensagem)



Atribui ao socket
- Endereço Internet (para receptor pode ser “any”, significando “recebe de qualquer IP”)
- Porta de comunicação (zero se atribuído dinamicamente pelo sistema)

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

int bind (int s, struct sockaddr *name, int namelen);

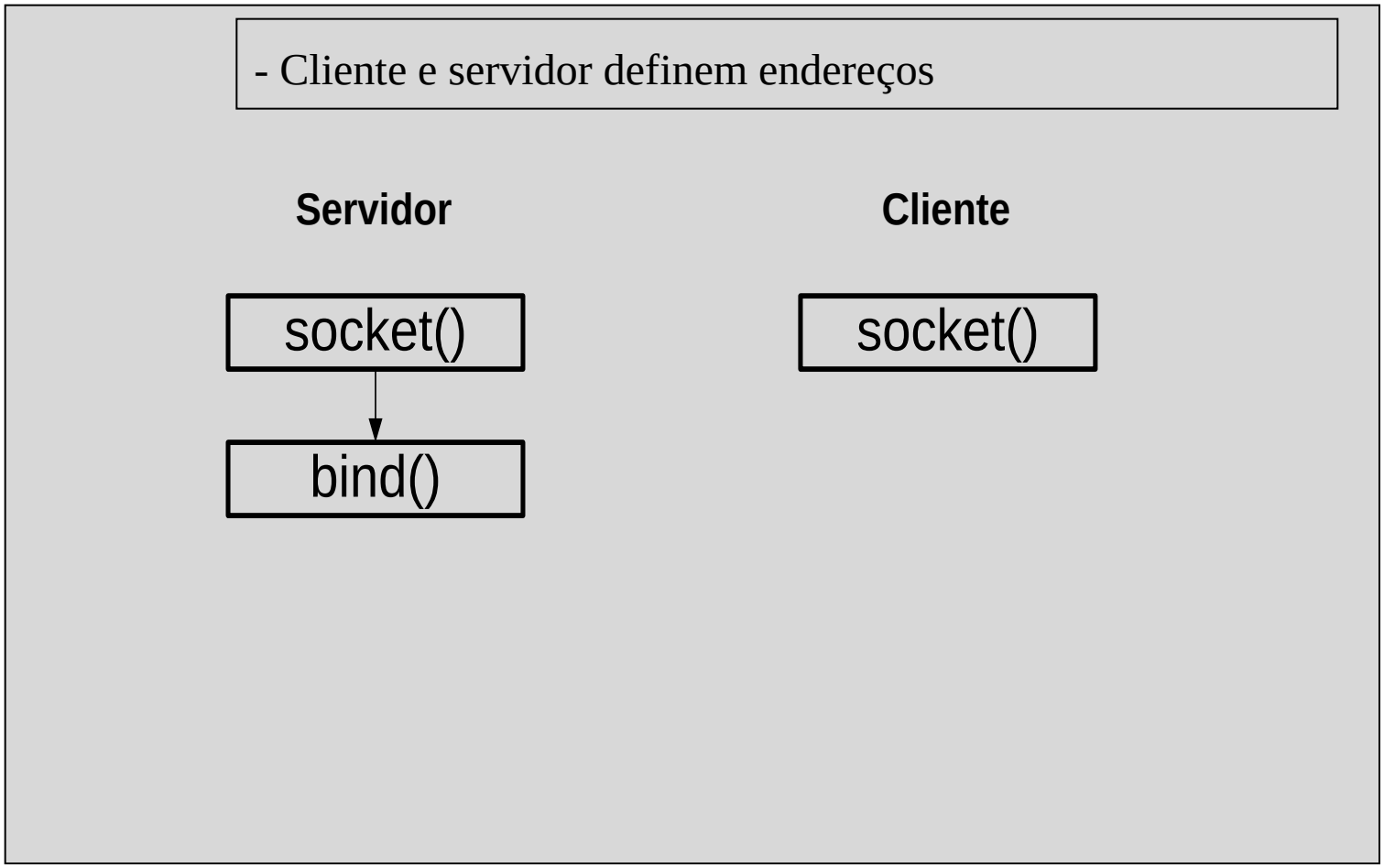
- int s
  - Valor retornado em socket()
- struct sockaddr *name
  - name.sin_family = AF_INET;
  - name.sin_addr.s_addr = INADDR_ANY; // recebe de qq IP; network order*
  - name.sin_port = htons(0); // 0 = atribuição dinâmica; network order*
- int namelen
  - Tamanho da estrutura “name” = (sizeof(name))

- * Ordenamento dos bytes para 0x102 (hexadecimal)
  - Default é ordenamento de “host” (“byte order”): primeiro MSB: “01 02”
  - Ordenamento de “network”: último MSB: “02 01”. Htons = “Host to Network Short”

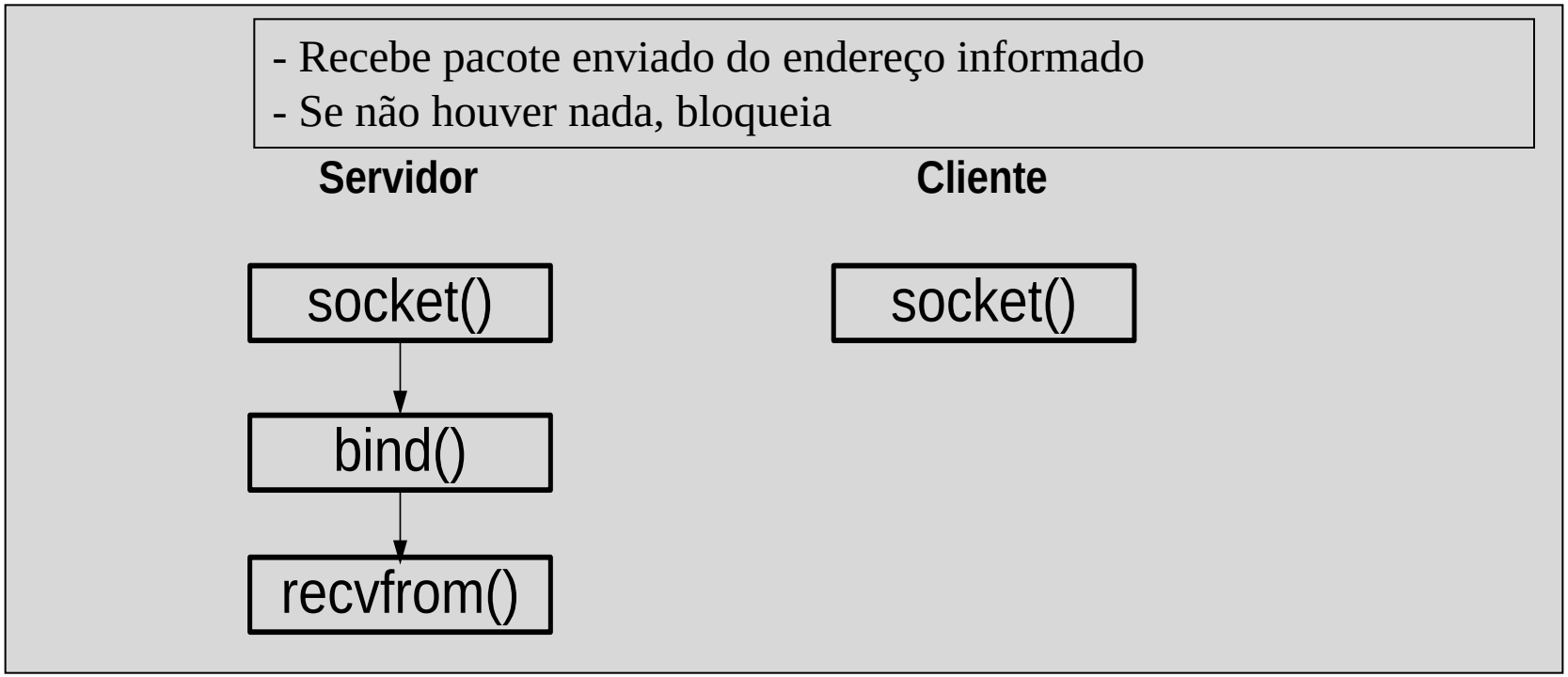
Exemplo

```
struct  sockaddr_in peer;
SOCKET  s;
int porta, peerlen;

...
memset((void *)&peer, 0, sizeof(struct sockaddr_in)); // zera estrutura
peer.sin_family = AF_INET;
peer.sin_addr.s_addr = htonl(INADDR_ANY); // Recebe de qualquer IP
peer.sin_port = htons(porta); // Recebe na porta especificada na linha de comando
peerlen = sizeof(peer);

// Associa socket com estrutura peer
if(bind(s,(struct sockaddr *) &peer, peerlen)) {
            printf("Erro no bind\n");
            exit(1);
}
```

Comunicação via UDP



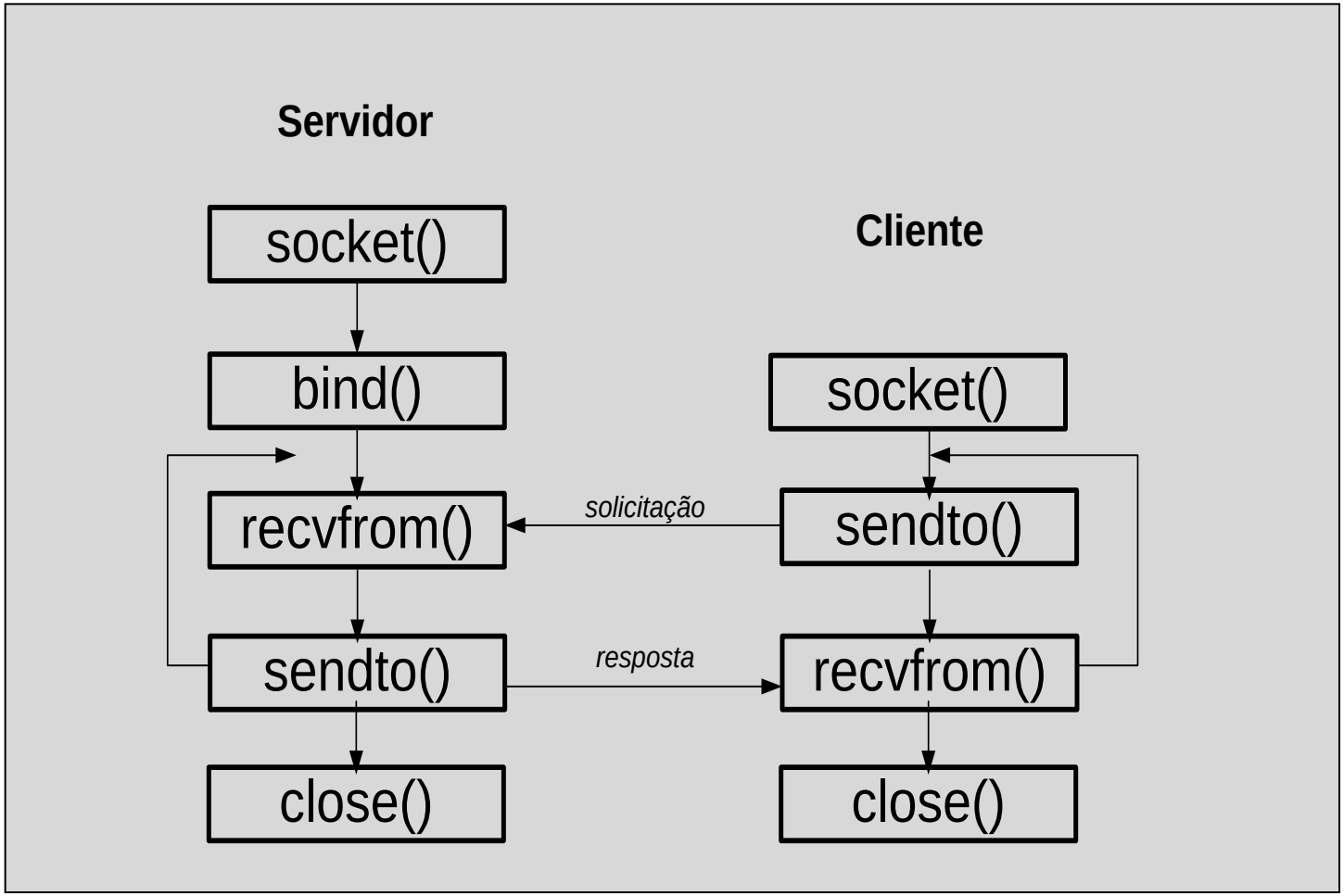
Comunicação via UDP



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

OBS: No recvfrom a estrutura *from é atualizada para conter o IP do transmissor. Dessa forma já se sabe para quem enviar a resposta.

Comunicação via UDP – Diagrama completo

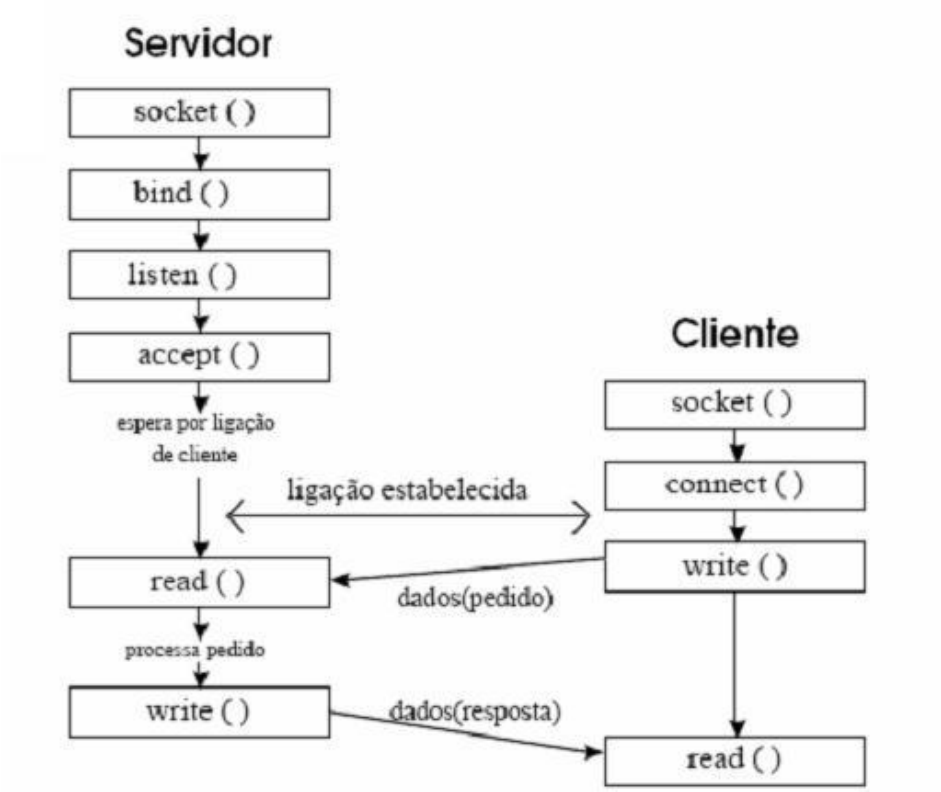


Exercício

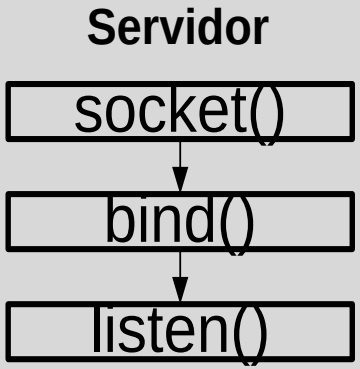
- O aluno deve enviar no relatório o seguinte:
  - Documento explicando o algoritmo utilizado, a linha de comando para uso dos programas e imagens mostrando o resultado de sua execução. Essas imagens podem ser obtidas através do programa "DU Meter" ou "Netmeter" ou "Netpersec" ou pelo gerenciador de tarefas ou outro método equivalente.
  - Códigos EXECUTÁVEIS;
  - Códigos FONTE.
- (4 pts) Verificar as estruturas analisadas no diretório "esqueleto_udp". Testar programa como está. Transformar o transmissor para enviar um número bits/s na rede de acordo com as seguintes linhas de comando:
  - "-r *n*": onde n = multiplicação da idade da dupla em kbit/s. Por exemplo, se os dois possuem 20 anos, deve ser 400 kbit/s.
  - "-r 1000": 1 Mbit/s
  - "-r 2000": 2Mbit/s

Sockets TCP

- Sockets TCP necessitam uma “conexão” antes da troca de dados
- Servidor se prepara para receber (listen) antes de começar a receber (accept)
- Cliente se conecta (connect) antes de transmitir
- Troca de dados se dá por “write” e “read”



Comunicação via TCP

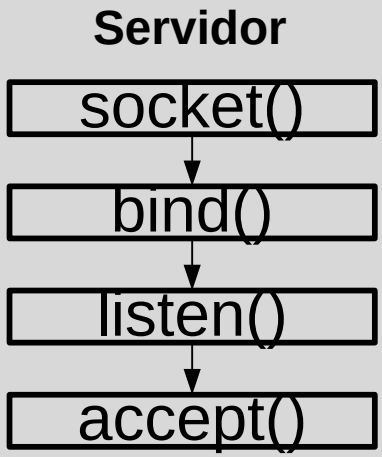


Listen declara:
- Que está pronto para receber conexões
- Até quantas consultas podem ser enfileiradas, ou seja, o comprimento da fila de espera de novos pedidos

- int sockfd
  - Valor retornado por socket()
- int backlog
  - comprimento da fila de espera de novos pedidos de ligação.

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

Comunicação via TCP – Accept

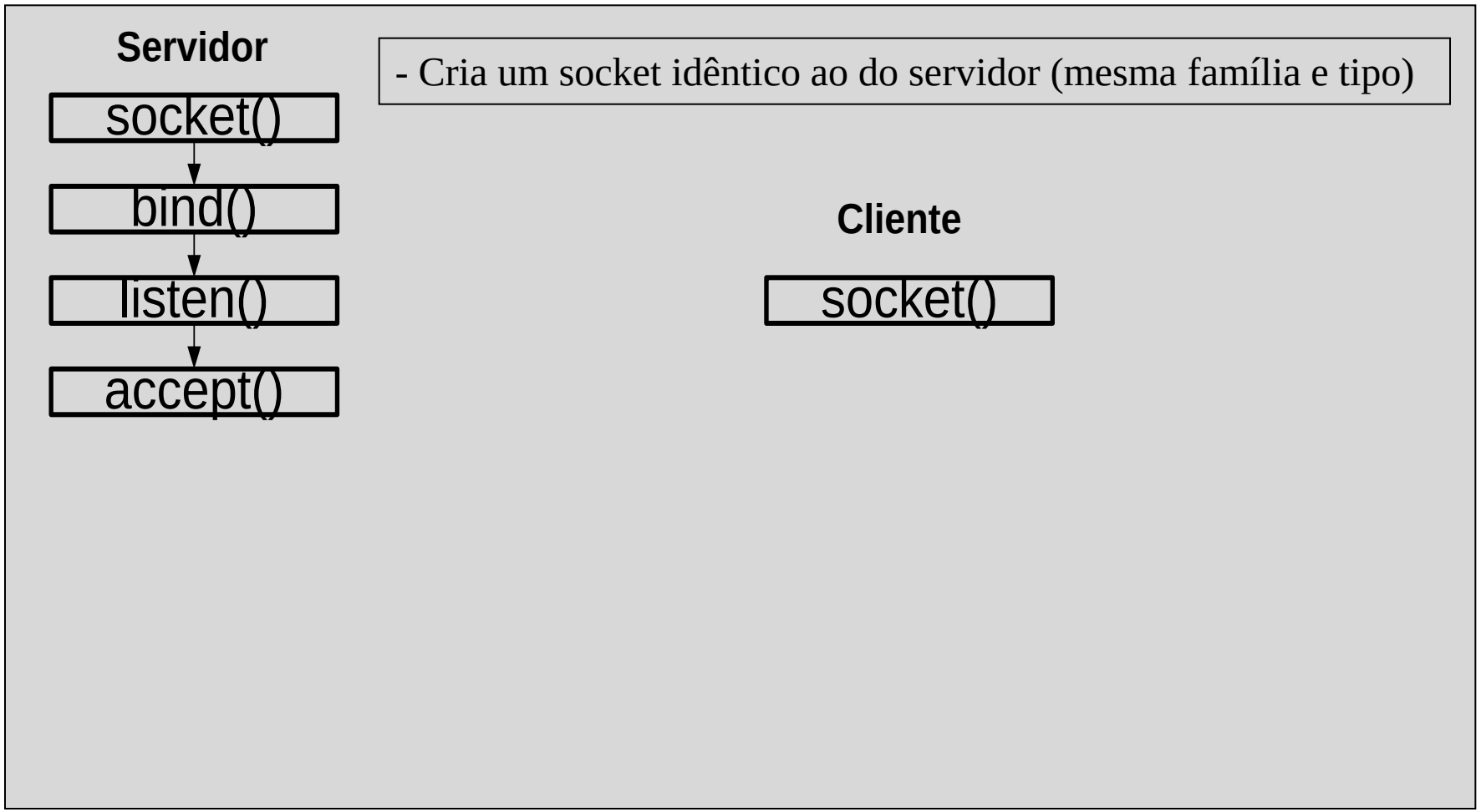


Accept:
- Bloqueia até que haja pedido de conexão
- Quando houver algum, aceita

- int sockfd
  - Valor retornado por socket()
- struct sockaddr *peer
  - Recebe o endereço da conexão
- int *peerlen
  - Tamanho (sizeof) da struct peer

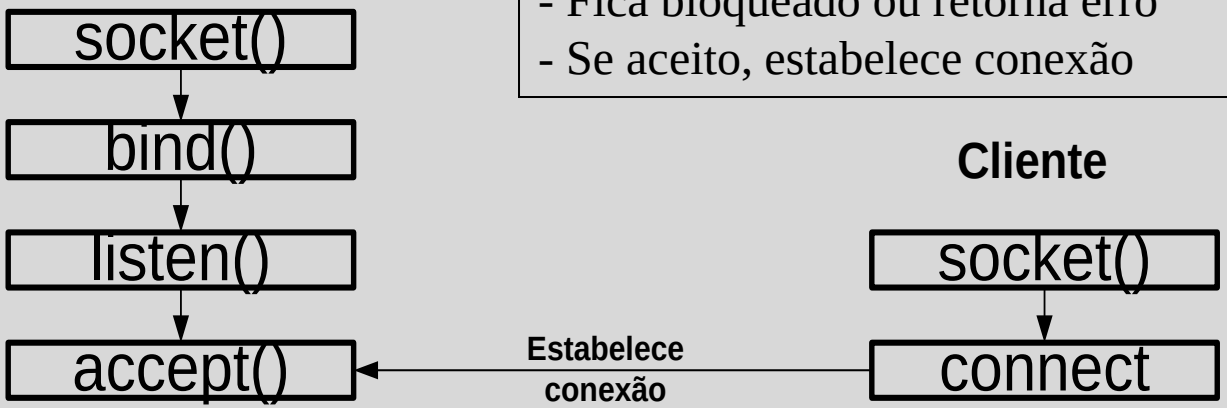
newsock = (int) **accept** (int **sockfd**, struct sockaddr ***peer**, int ***peerlen**)

Comunicação via TCP – Cliente



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

TCP necessita conexão – Connect

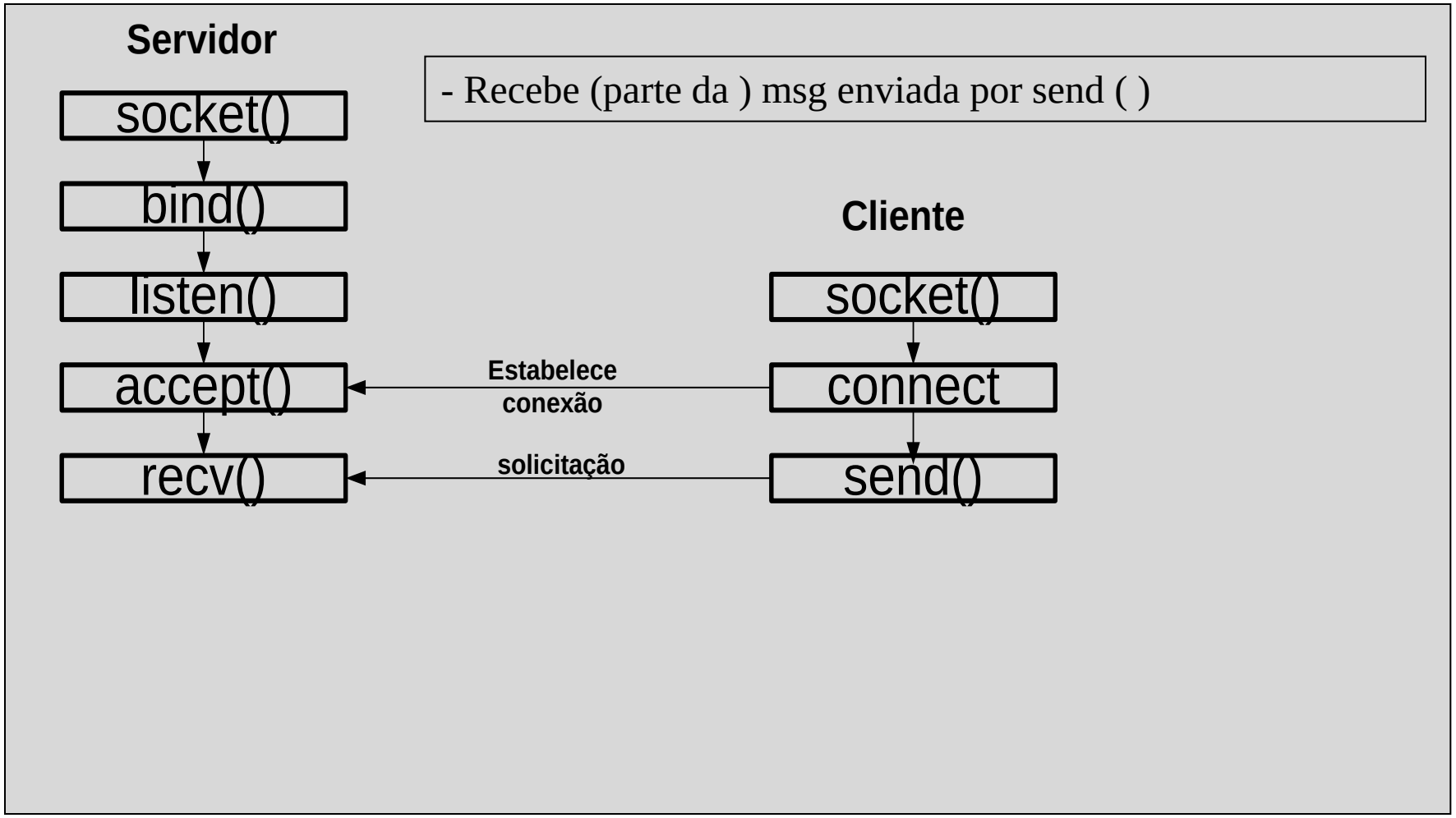


- Pede uma conexão ao servidor
- Fica bloqueado ou retorna erro
- Se aceito, estabelece conexão

- struct sockaddr *servaddr
  - Dados para quem irá se conectar.
- int peer_len
  - Tamanho (sizeof) de peer

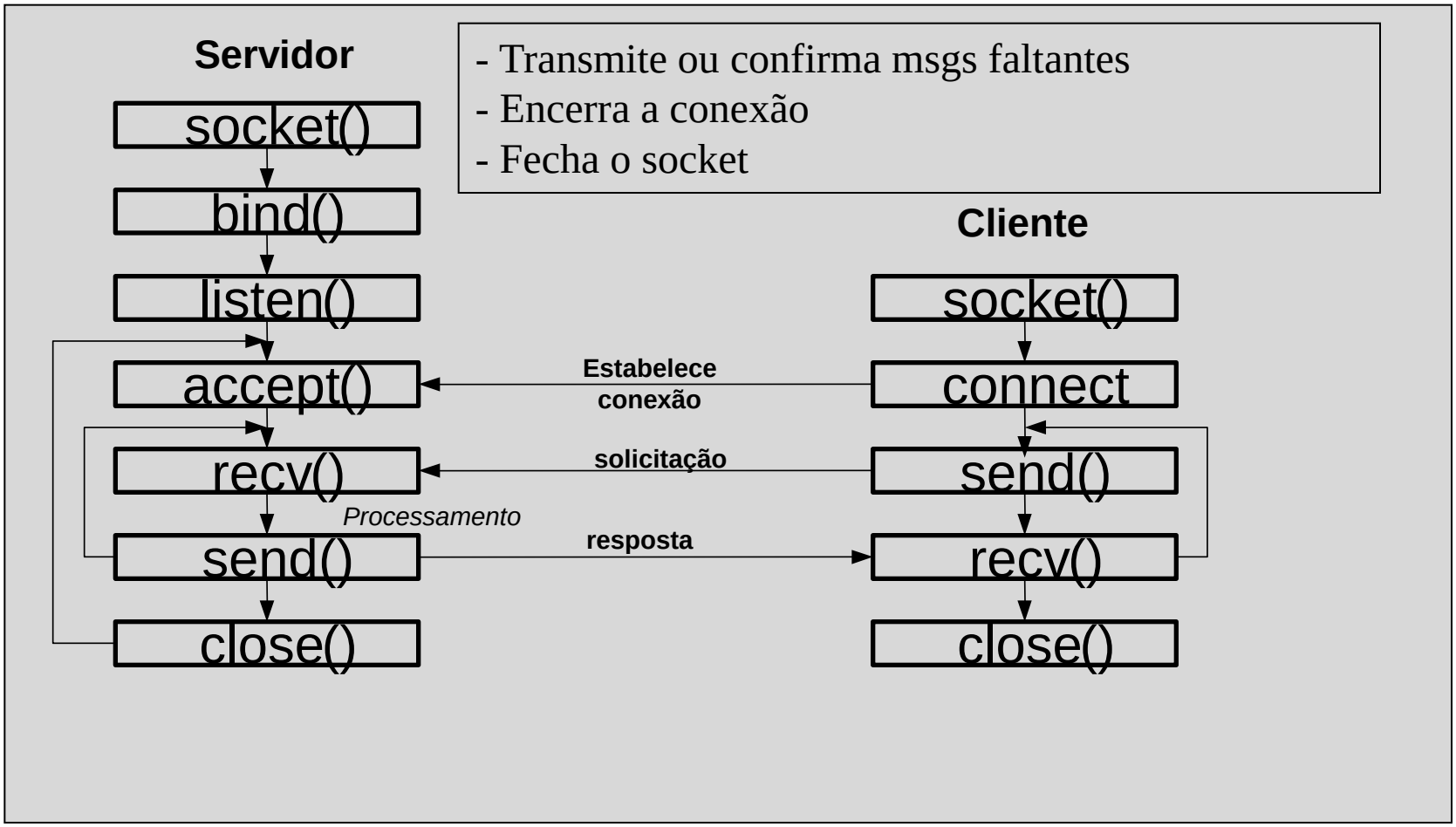
ret = (int) **connect** (int **sockfd**, struct sockaddr ***servaddr**, int **peerlen**)

Comunicação via TCP – Recv e Send



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

Comunicação via TCP – Diagrama completo



ret = (int) **close** (int **sockfd**)

Exercícios

- Valem mesmas dicas de relatório da semana passada (Explicar algoritmo, mostrar as imagens, ...)
- (4 pts) Verificar as estruturas analisadas no diretório “esqueleto_tcp”. Testar programa como está. Analisar eqüidade do tráfego na rede, alterando o programa para ficar enviando dados na máxima velocidade possível, e testando com várias transmissões simultâneas (mínimo de 2 conexões –dois servidores (duas instâncias do servidor em portas diferentes) e dois clientes). **Gerar log com a média de tráfego por segundo**. Pode-se realizar o relatório com duas máquinas, mas se o grupo quiser, poderá utilizar 3 máquinas ou mais (para ver na ferramenta de redes, além do log, a variação de carga na rede)
- Explicar (1 pt cada):
  - O que é equidade de tráfego e em qual protocolo ela foi observada nativamente no laboratório?
  - Qual o nível do modelo OSI para fazer controle de fluxo quando o programador utiliza TCP e quando utiliza UDP?

Exercícios (não precisa fazer)

Os exercícios podem ser efetuados com base nos esqueletos da página. Devem ser efetuados em linguagem C para que se visualize bem a utilização dos sockets.

- Sockets TCP: Criar programa cliente de http simples, ou seja, que conecte na porta 80 de um servidor http qualquer e efetue o comando “GET”. Entrada: endereço web do servidor http. Saída: página inicial obtida com o comando GET.
- Sockets UDP: Criar programa de chat entre dois ou mais computadores via UDP. Cada usuário deve ter condições de enviar mensagens. Na entrada, o usuário deve digitar um “apelido” que deve aparecer em cada mensagem sua. As mensagens devem ser enviadas a todos.