

Introdução a Programação de Sockets em C usando TCP/IP

Professor: Panagiota Fatourou
AT: Eleftherios Kosmas

CSD - Maio de 2012

Introdução

Rede de Computadores

hosts, roteadores,
canais de comunicação

Os hosts executam aplicativos

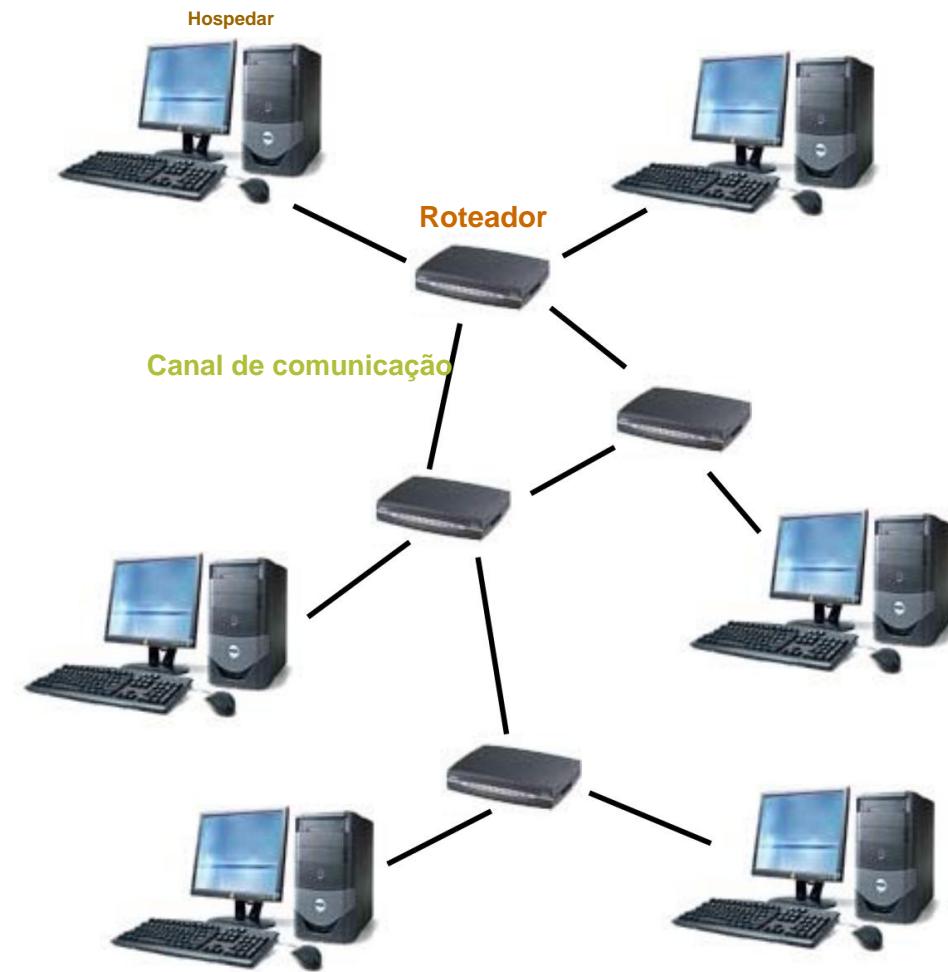
Os roteadores encaminham informações

Pacotes: sequência de bytes

contém informações de controle
por exemplo host de destino

Protocolo é um acordo

significado de pacotes
estrutura e tamanho dos pacotes
por exemplo Protocolo de Transferência de
Hipertexto (HTTP)



Famílias de Protocolos - TCP/IP

Vários protocolos para diferentes problemas

Conjuntos de protocolos ou famílias de protocolos: TCP/IP

O TCP/IP fornece conectividade **de ponta a ponta**, especificando como os dados devem ser

formatado,

endereçado,

transmitido,

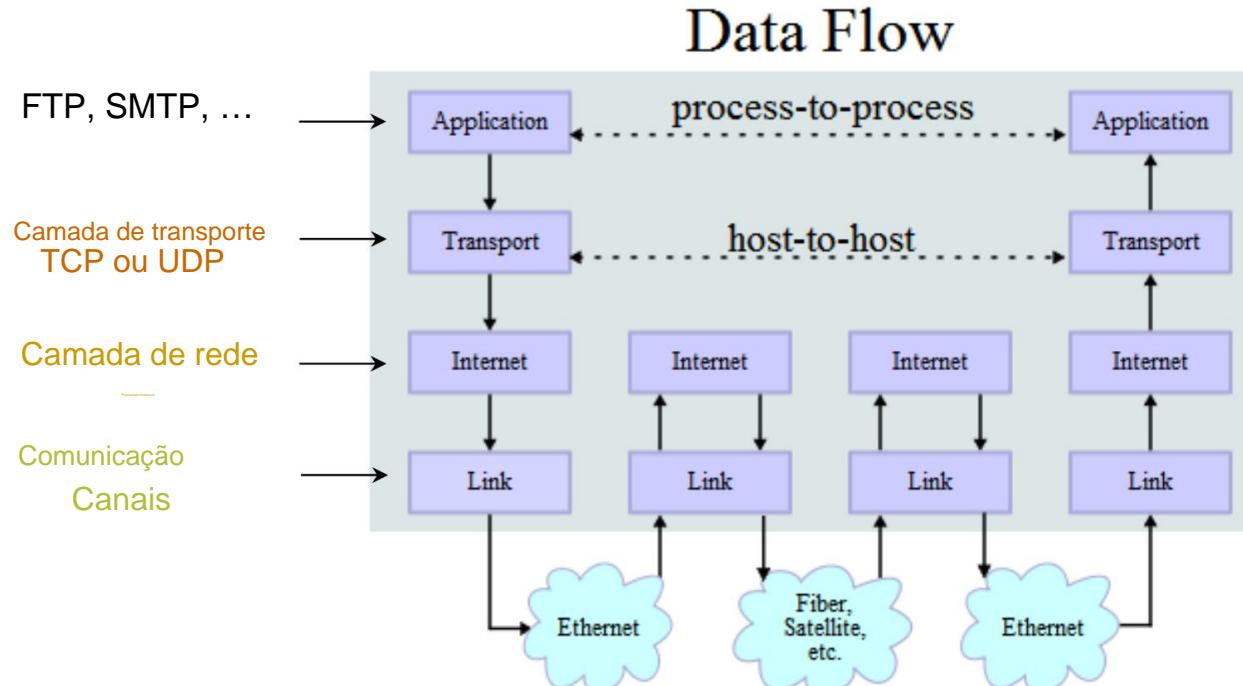
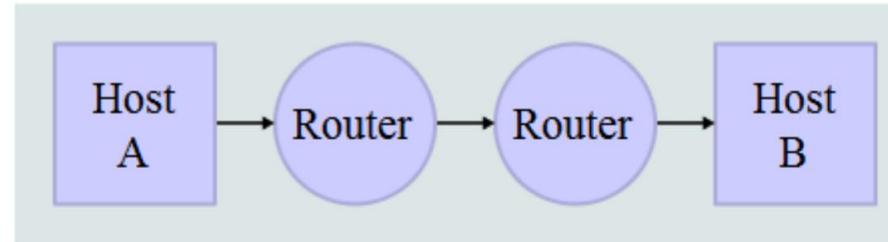
roteado e

recebido no destino

pode ser usado na internet e em redes privadas autônomas

é organizado em **camadas**

TCP/IP Network Topology



* a imagem foi retirada de "http://en.wikipedia.org/wiki/TCP/IP_model"

Protocolo de Internet (IP)

fornecer um serviço **de datagrama**

os pacotes são manipulados e entregues de forma independente

protocolo **de melhor esforço**

pode perder, reordenar ou duplicar pacotes

cada pacote deve conter um **endereço IP** de seu destino

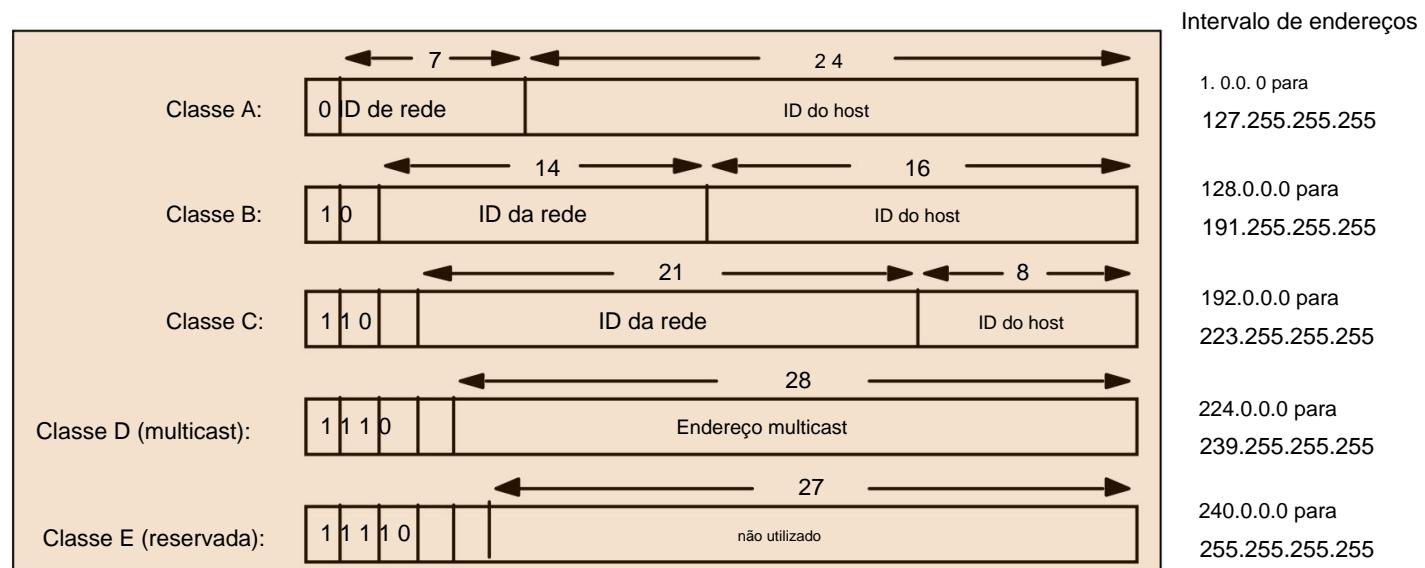


Endereços - IPv4

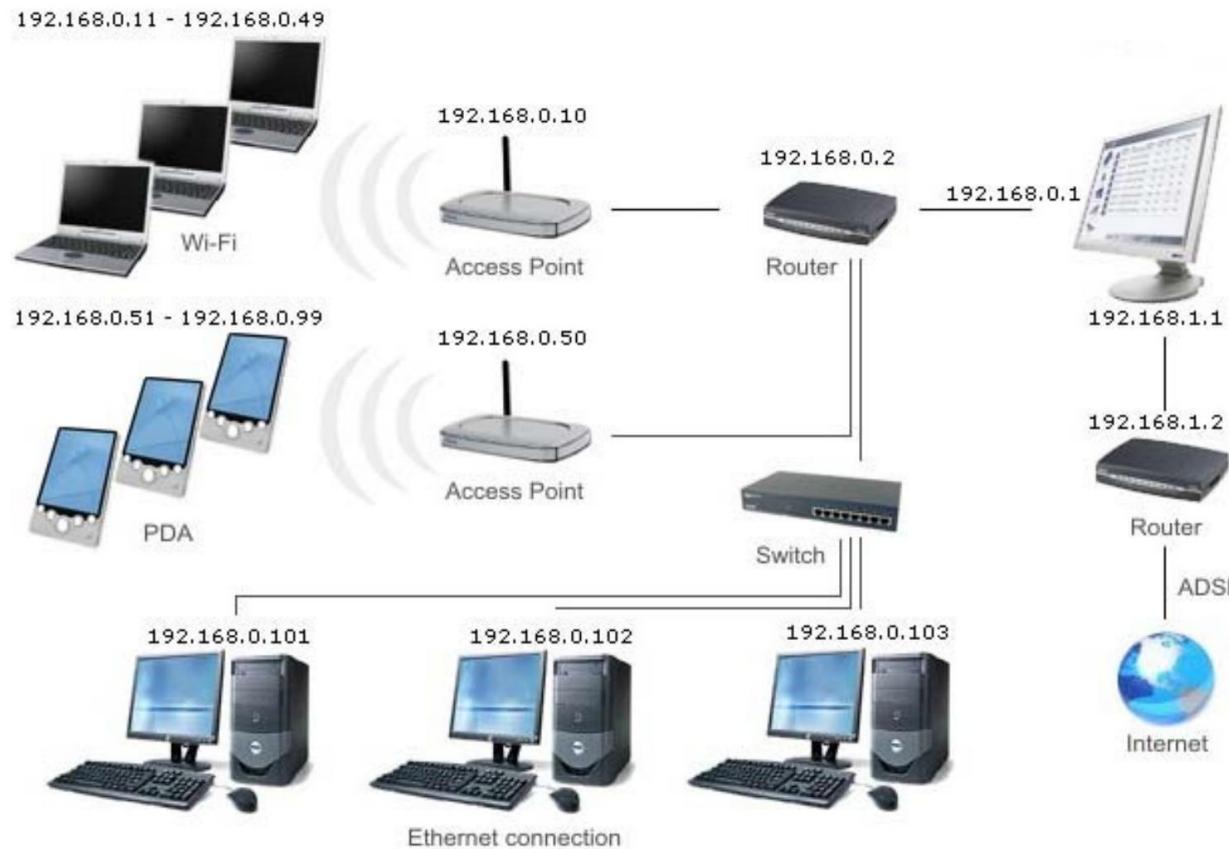
Os 32 bits de um endereço IPv4 são divididos em **4 octetos**, ou campos de 8 bits (valor de 0 a 255 em notação decimal).

Para redes de tamanhos diferentes,

o primeiro (para redes grandes) a três (para redes pequenas) octetos podem ser usados para identificar a **rede**, enquanto o restante dos octetos pode ser usado para identificar o **nó** na rede.



Endereços de rede local - IPv4



TCP vs UDP

Ambos usam **números de porta**, construção específica do aplicativo que serve como um ponto de extremidade de comunicação, inteiro sem sinal de 16 bits, variando de 0 a 65535 para fornecer transporte **de ponta a ponta**

UDP: Protocolo de Datagrama do Usuário

sem confirmações, sem retransmissões

fora de ordem, duplica possíveis conexões

sem conexão, ou seja, o aplicativo indica o destino de cada pacote

TCP: Protocolo de Controle de Transmissão

canal de fluxo de bytes confiável (em ordem, todos chegam, sem duplicatas)
semelhante a E/S de arquivo

controle de fluxo

orientado para conexão

bidirecional

TCP vs UDP

O TCP é usado para serviços com grande capacidade de dados e uma conexão persistente

O UDP é mais comumente usado para pesquisas rápidas e ações de consulta e resposta de uso único.

Alguns exemplos comuns de TCP e UDP com suas portas padrão:

Pesquisa DNS	UDP 53
FTP	TCP 21
HTTP	TCP 80
POP3	TCP 110
Telnet	TCP 23

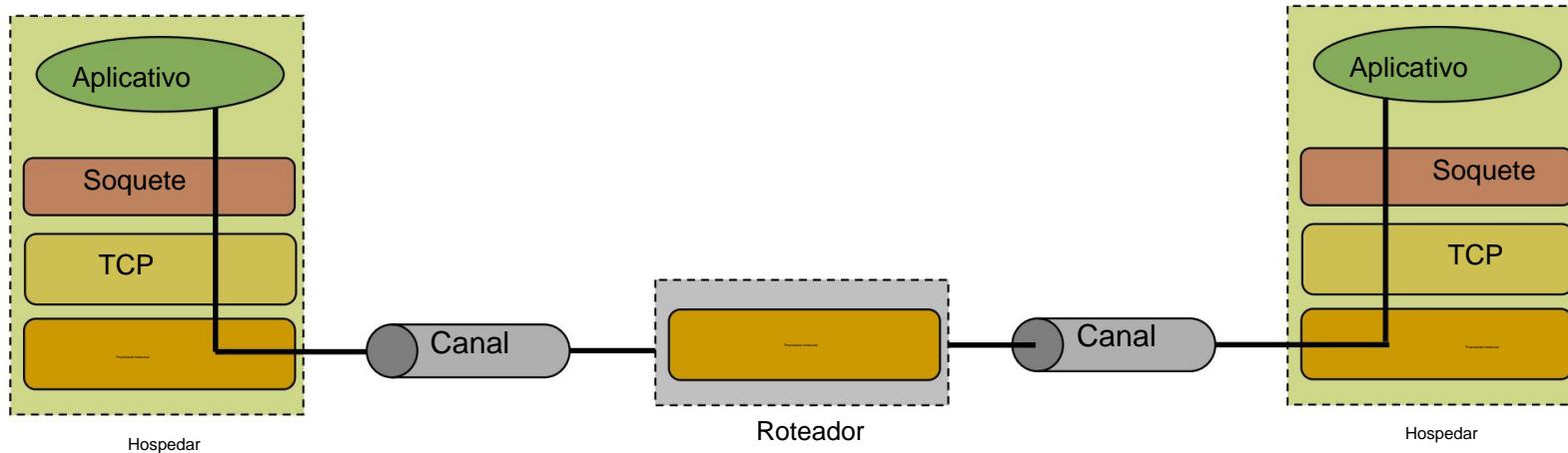
Soquetes Berkley

Universalmente conhecido como **Sockets**

É uma abstração através da qual uma aplicação pode enviar e receber dados

Fornecer **acesso genérico** aos serviços de comunicação entre processos

por exemplo IPX/SPX, Appletalk, TCP/IP API padrão para rede



Soquetes

Identificado exclusivamente por um

endereço de internet

um protocolo de ponta a ponta (por exemplo, TCP ou UDP) um número

de porta Dois tipos de

soquetes (TCP/IP)

Soquetes **de fluxo** (por exemplo, usa TCP)

fornecer serviço de fluxo de bytes confiável

Soquetes **de datagrama** (por exemplo, usa UDP)

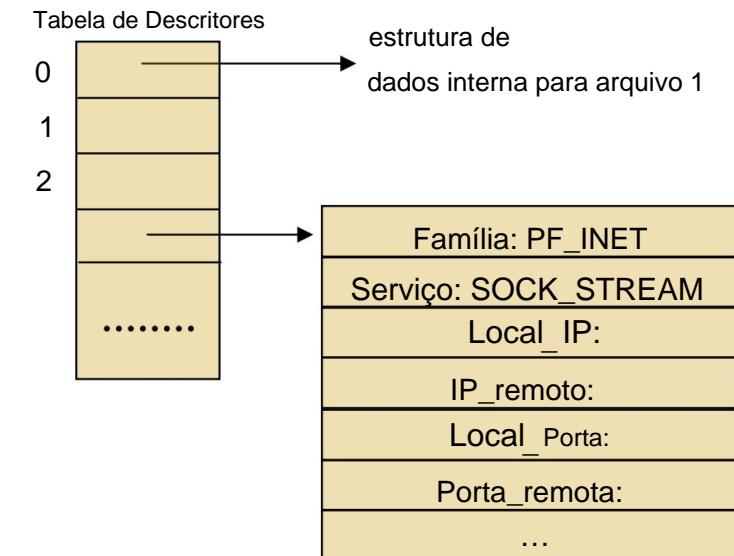
fornecer mensagens de serviço de datagrama de melhor

esforço de até 65.500 bytes Socket estende

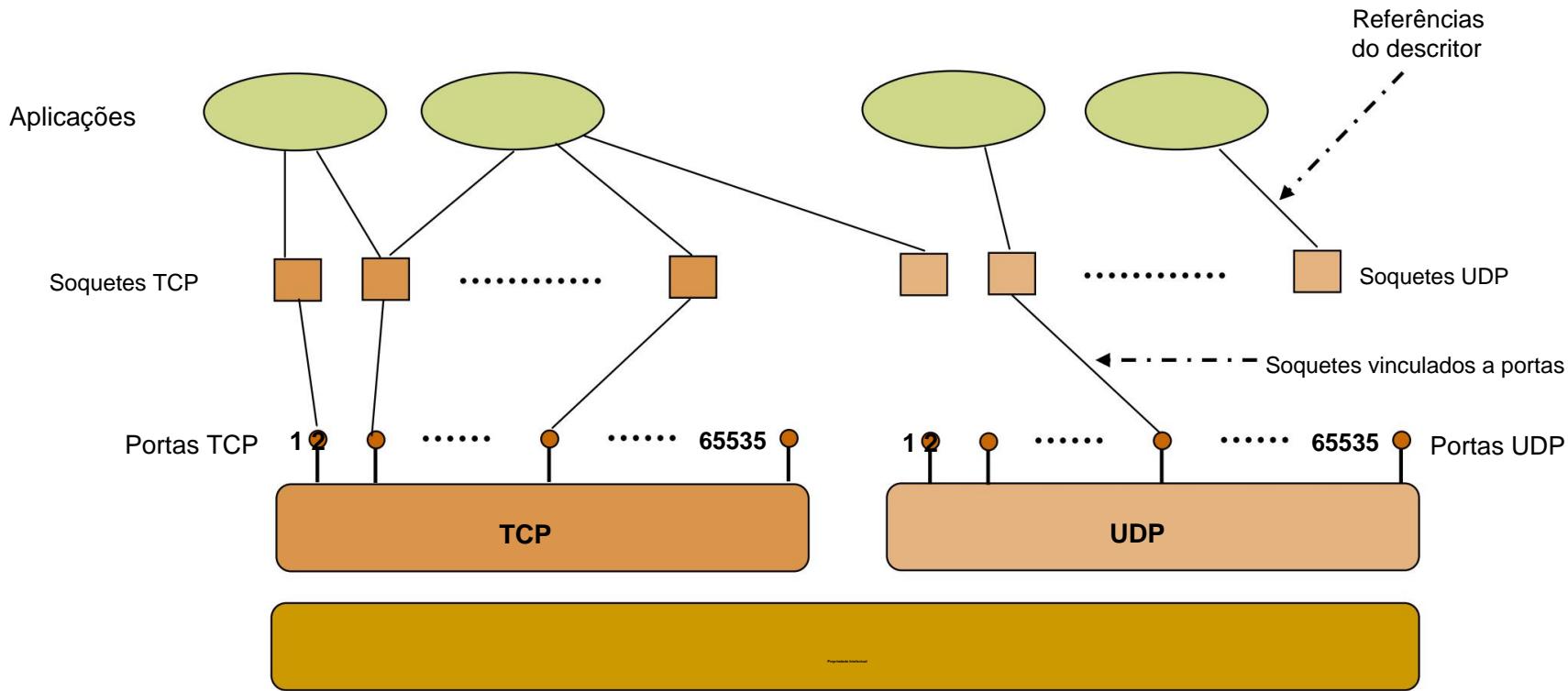
os recursos de E/S UNIX convencionais

descritores de arquivo para comunicação de rede estenderam as

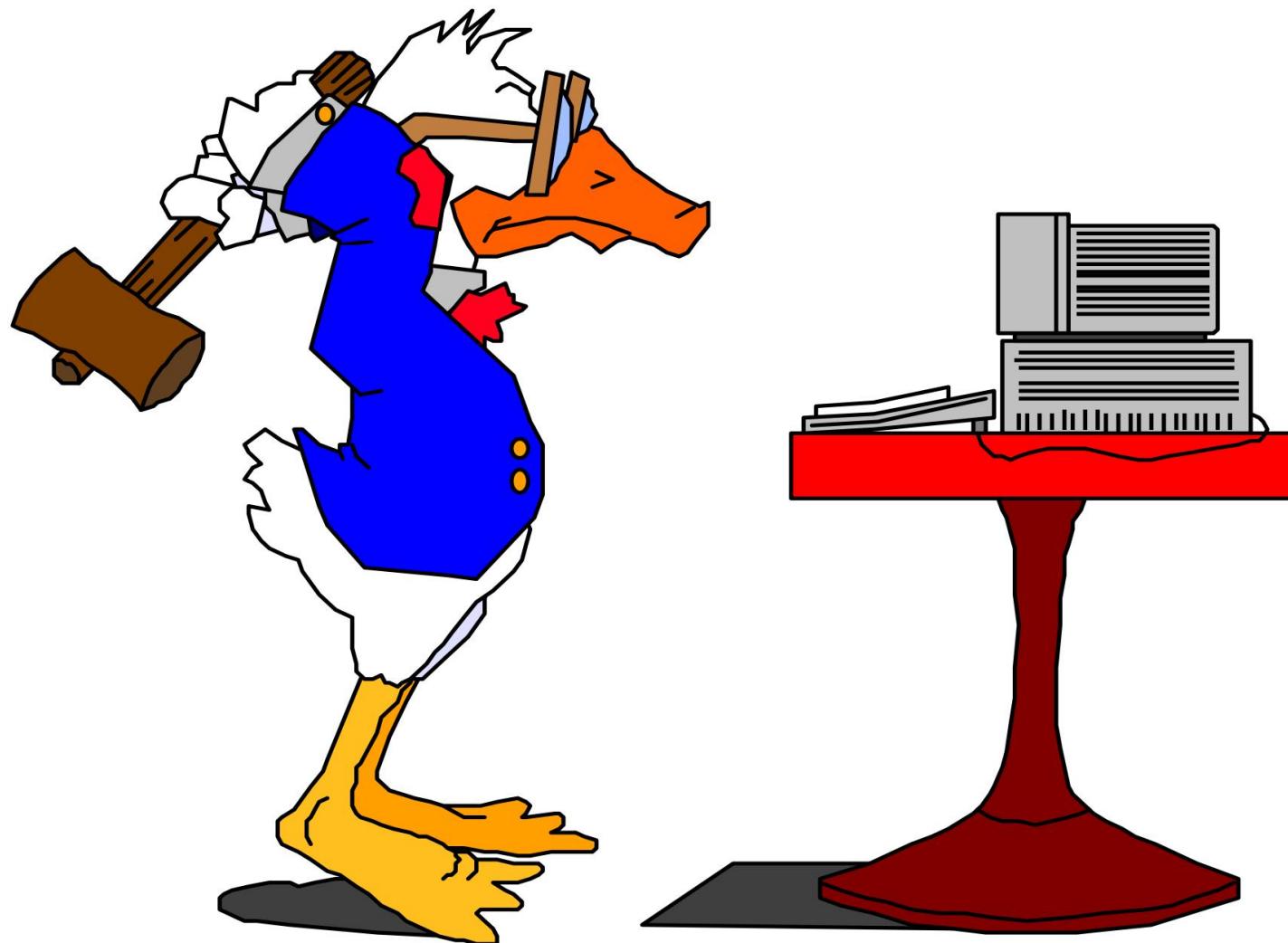
chamadas de sistema de leitura e gravação



Soquetes



Programação de Sockets



Comunicação cliente-servidor

Servidor

espera passivamente e responde aos clientes

soquete **passivo**

Cliente

inicia a comunicação

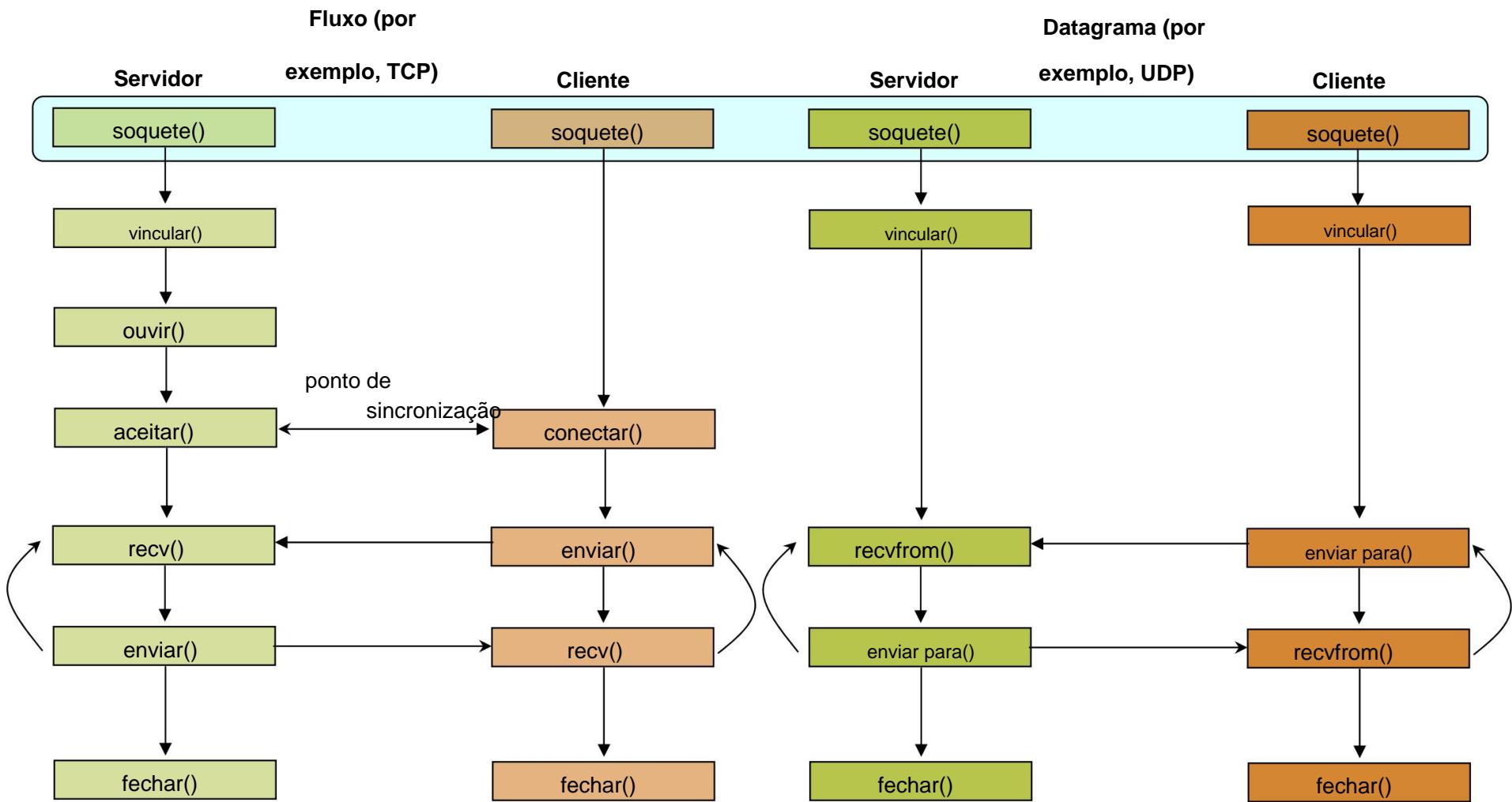
deve saber o endereço e a porta do servidor

soquete **ativo**

Sockets - Procedimentos

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Comunicação Cliente-Servidor - Unix



Criação de soquete em C: socket()

int sockid = socket(família, tipo, protocolo);

sockid: descritor de socket, um inteiro (como um identificador de arquivo)

família: inteiro, domínio de comunicação, por exemplo,

PF_INET, protocolos IPv4, endereços de Internet (normalmente usados)

PF_UNIX, Comunicação local, Endereços de arquivo

tipo: tipo de comunicação

SOCK_STREAM - serviço confiável, bidirecional e baseado em conexão

SOCK_DGRAM - não confiável, sem conexão, mensagens de comprimento máximo

protocolo: especifica o protocolo

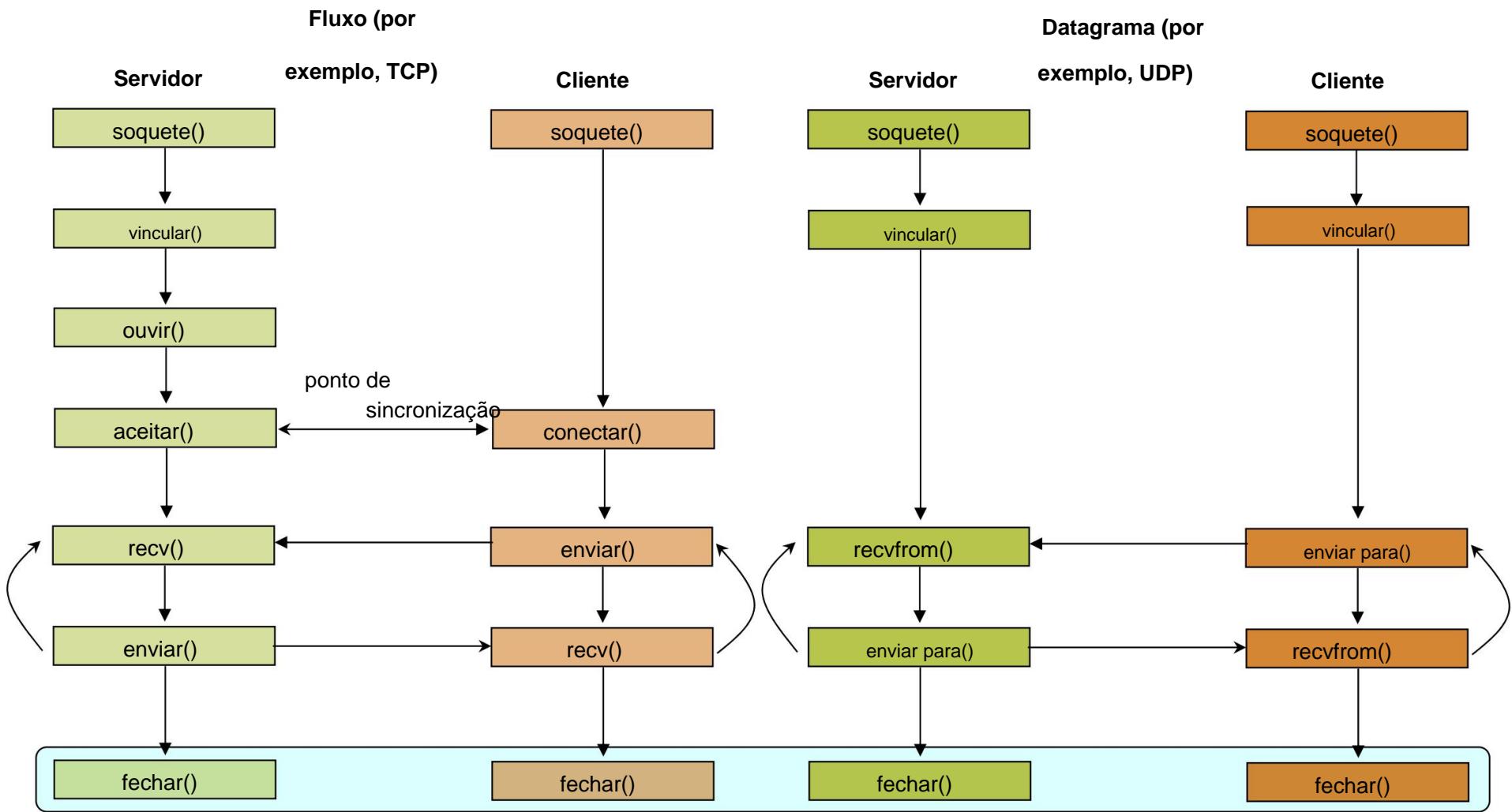
IPPROTO_TCP IPPROTO_UDP

geralmente definido como 0 (ou seja, usar protocolo padrão)

em caso de falha retorna -1

NOTA: a chamada de soquete não especifica de onde os dados virão, nem para onde irão – ela apenas cria a interface!

Comunicação Cliente-Servidor - Unix



Fechamento de soquete em C: close()

Quando terminar de usar um soquete, o soquete deve ser fechado

status = close(sockid); sockid: o

descritor do arquivo (socket sendo fechado) **status:** 0
se bem-sucedido, -1 se erro

Fechar um socket

fecha uma conexão (para stream socket)
libera a porta usada pelo socket

Especificando Endereços

A API de soquete define um tipo de dado **genérico** para endereços:

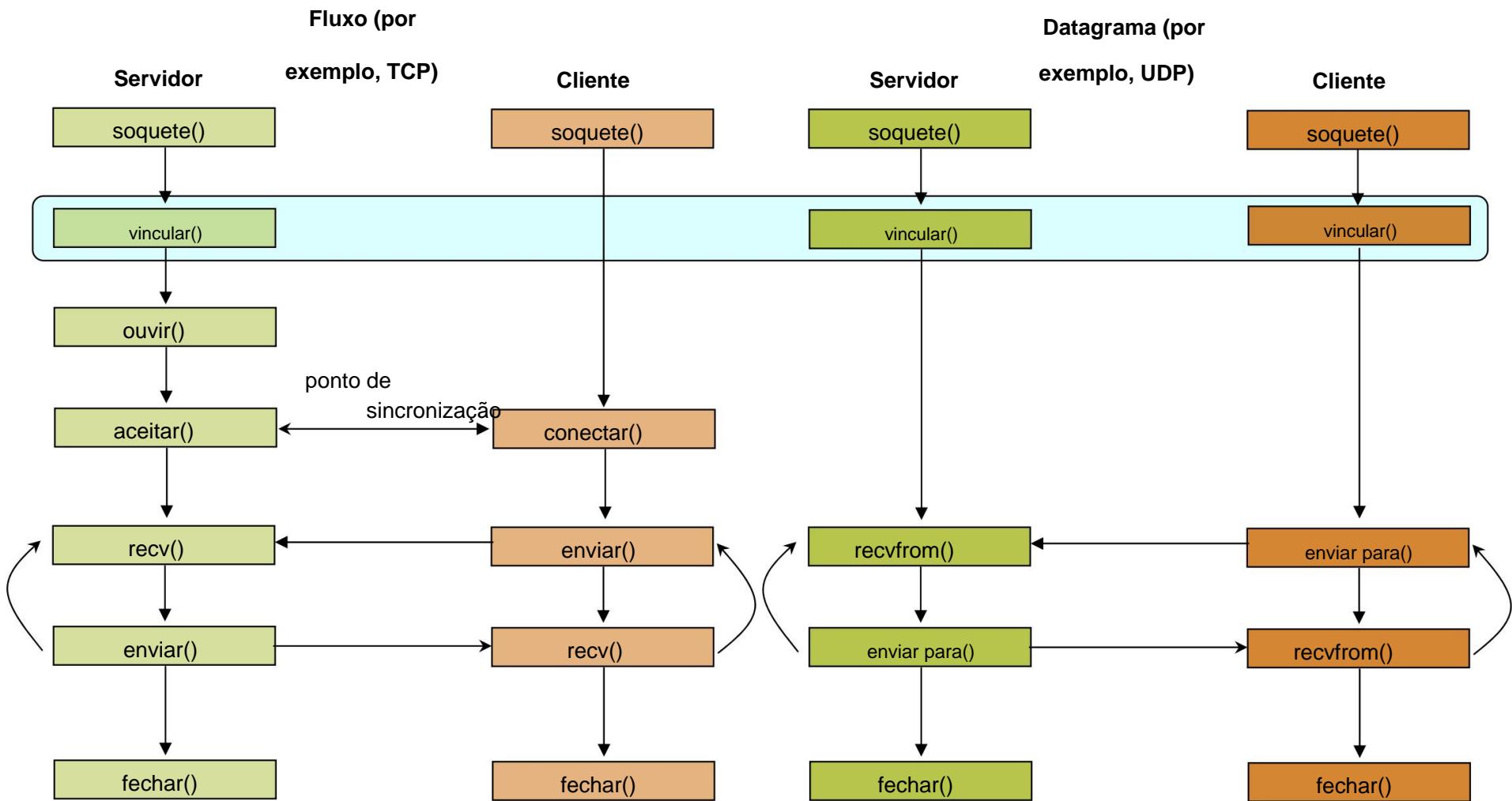
```
estrutura sockaddr {  
    unsigned short sa_family; /* Família de endereços (por exemplo, AF_INET) */  
    char sa_data[14]; /* Informações de endereço específicas da família */  
}
```

Forma particular do sockaddr usado para endereços **TCP/IP** :

```
estrutura in_addr {  
    s_addr longo sem sinal; /* Endereço de internet (32 bits) */  
}  
  
estrutura sockaddr_in {  
    sin_family curto não assinado; /* Protocolo de Internet (AF_INET) */  
    sin_port curto não assinado; struct /* Porta de endereço (16 bits) */  
    in_addr sin_addr; /* Endereço de internet (32 bits) */  
    caracteres sin_zero[8]; /* Não utilizado */  
}
```

Importante: sockaddr_in pode ser convertido para um sockaddr

Comunicação Cliente-Servidor - Unix



Atribuir endereço ao socket: bind()

associa e reserva uma porta para uso pelo soquete

```
int status = bind(sockid, &addrport, tamanho);
```

sockid: inteiro, descritor de soquete

addrport: struct sockaddr, o endereço (IP) e a porta da máquina

para servidor TCP/IP, o endereço de internet é geralmente definido como INADDR_ANY, ou seja, escolhe qualquer interface de entrada

tamanho: o tamanho (em bytes) da estrutura addrport

status: em caso de falha -1 é retornado

bind()- Exemplo com TCP

```
int meia;  
  
struct sockaddr_in endereço;  
sockid = soquete(PF_INET, SOCK_STREAM, 0);  
  
addrport.sin_family = AF_INET;  
addrport.sin_port = htons(5100);  
addrport.sin_addr.s_addr = htonl(INADDR_ANY);  
se(vincular(sockid, (struct sockaddr *) &addrport, tamanhode(addrport))!= -1) {  
...}
```

Ignorando o bind()

bind pode ser ignorado para ambos os tipos de sockets

Soquete de datagrama:

se apenas enviando, não há necessidade de vincular. O SO encontra uma porta cada vez que o soquete envia um pacote

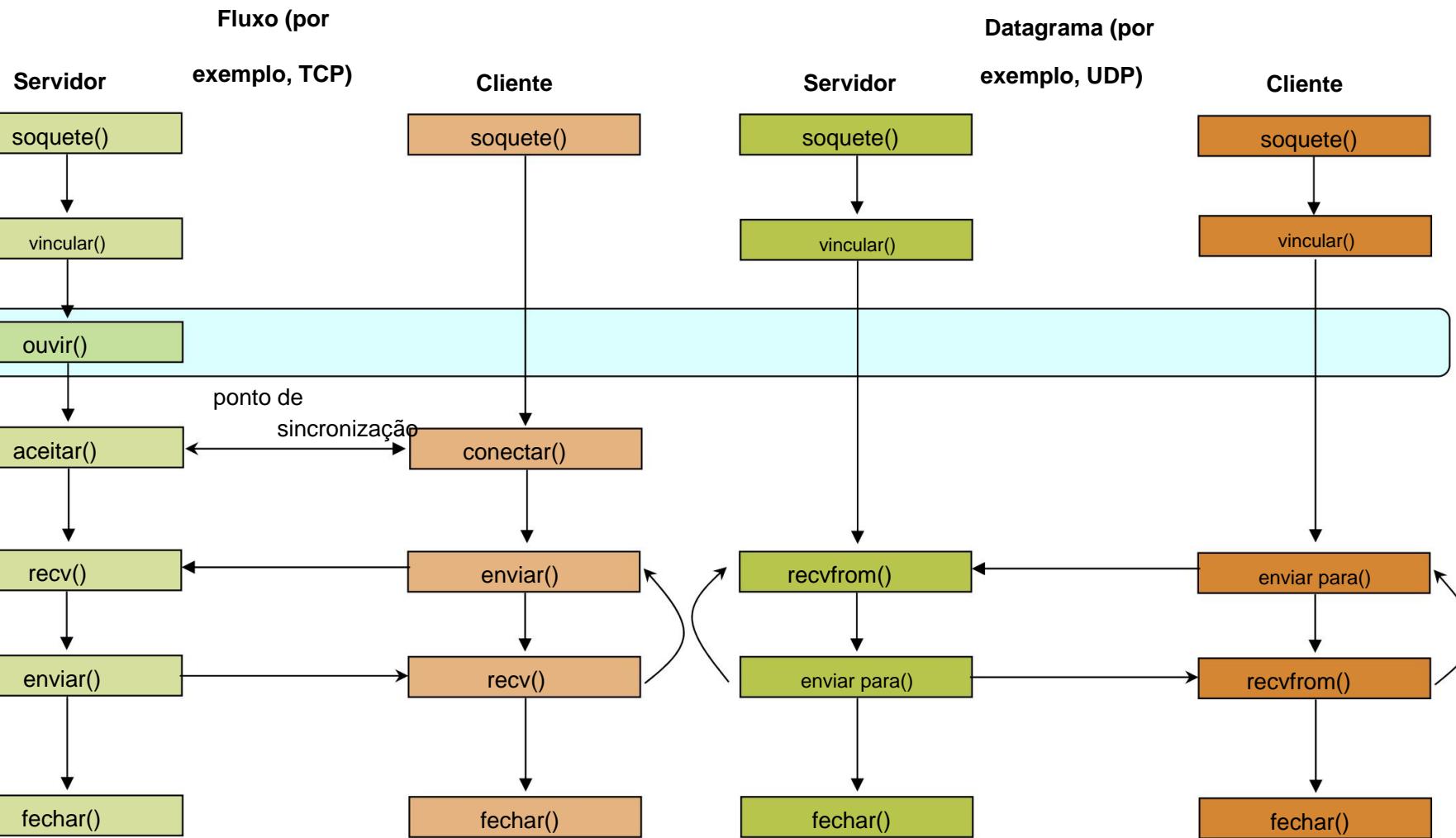
se receber, precisa vincular

Soquete de transmissão:

destino determinado durante a configuração da conexão

não precisa saber a porta de envio (durante a configuração da conexão, o destinatário é informado da porta)

Comunicação Cliente-Servidor - Unix



Atribuir endereço ao socket: bind()

Instrui a implementação do protocolo TCP a escutar conexões

```
int status = escutar(sockid, queueLimit);
```

sockid: inteiro, descriptor de soquete

queueLen: inteiro, # de participantes ativos que podem “esperar” por uma conexão

status: 0 se estiver ouvindo, -1 se estiver com erro

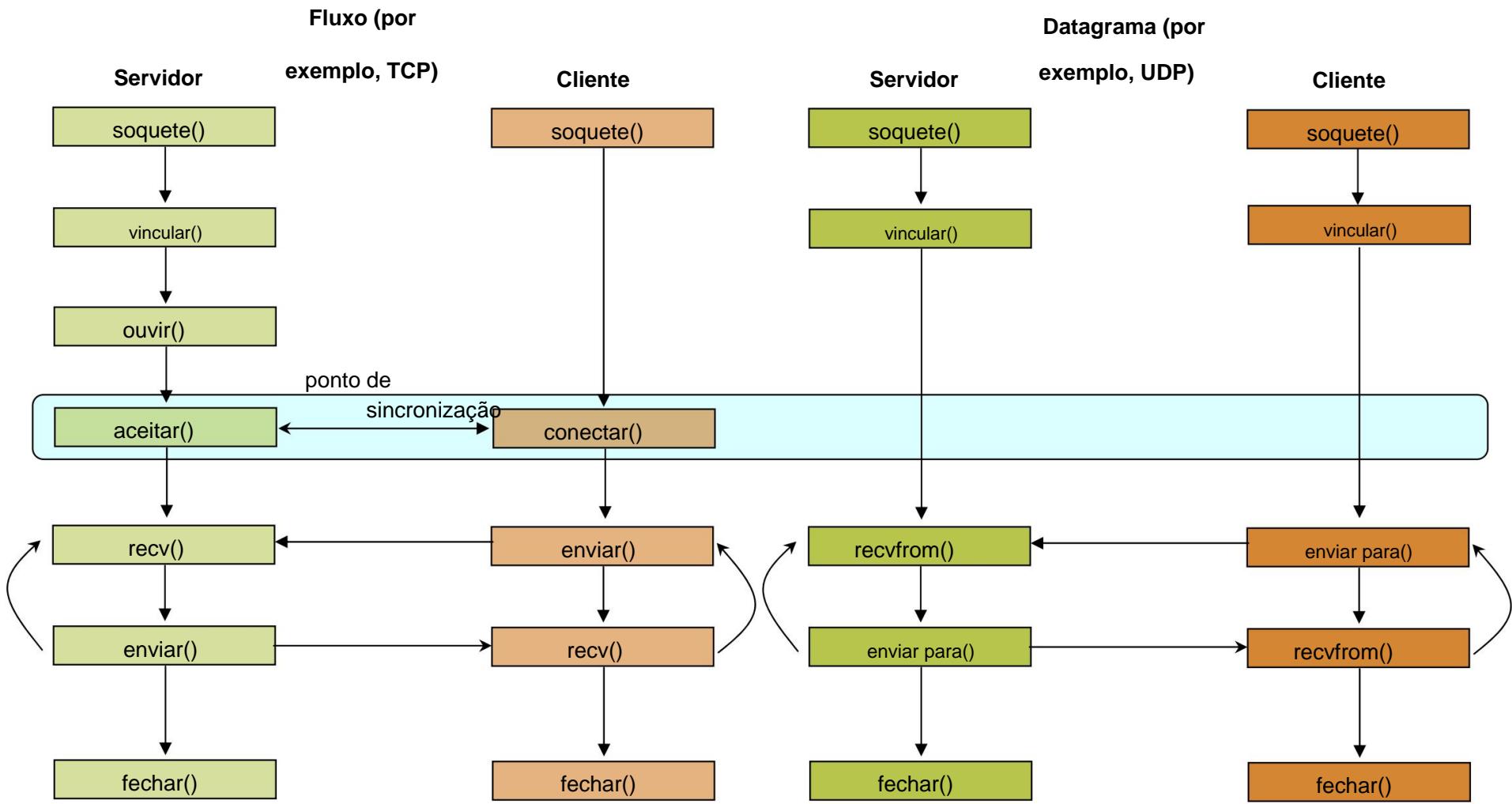
listen() não é bloqueante: retorna imediatamente

O soquete de escuta (sockid)

nunca é usado para enviar e receber

é usado pelo servidor apenas como uma forma de obter novos soquetes

Comunicação Cliente-Servidor - Unix



Estabelecer conexão: connect()

O cliente estabelece uma conexão com o servidor chamando connect()

```
int status = conectar(sockid, &estrangeiroAddr, addrlen);
```

sockid: inteiro, soquete a ser usado na conexão

foreignAddr: struct sockaddr: endereço do participante passivo

addrlen: inteiro, sizeof(nome)

status: 0 se a conexão for bem-sucedida, -1 caso contrário

connect() está bloqueando

Conexão de entrada: accept()

O servidor obtém um soquete para uma conexão de cliente de entrada chamando accept()

```
int s = aceitar(sockid, &clientAddr, &addrLen);
```

s: inteiro, o novo soquete (usado para transferência de dados)

sockid: inteiro, o soquete original (sendo escutado)

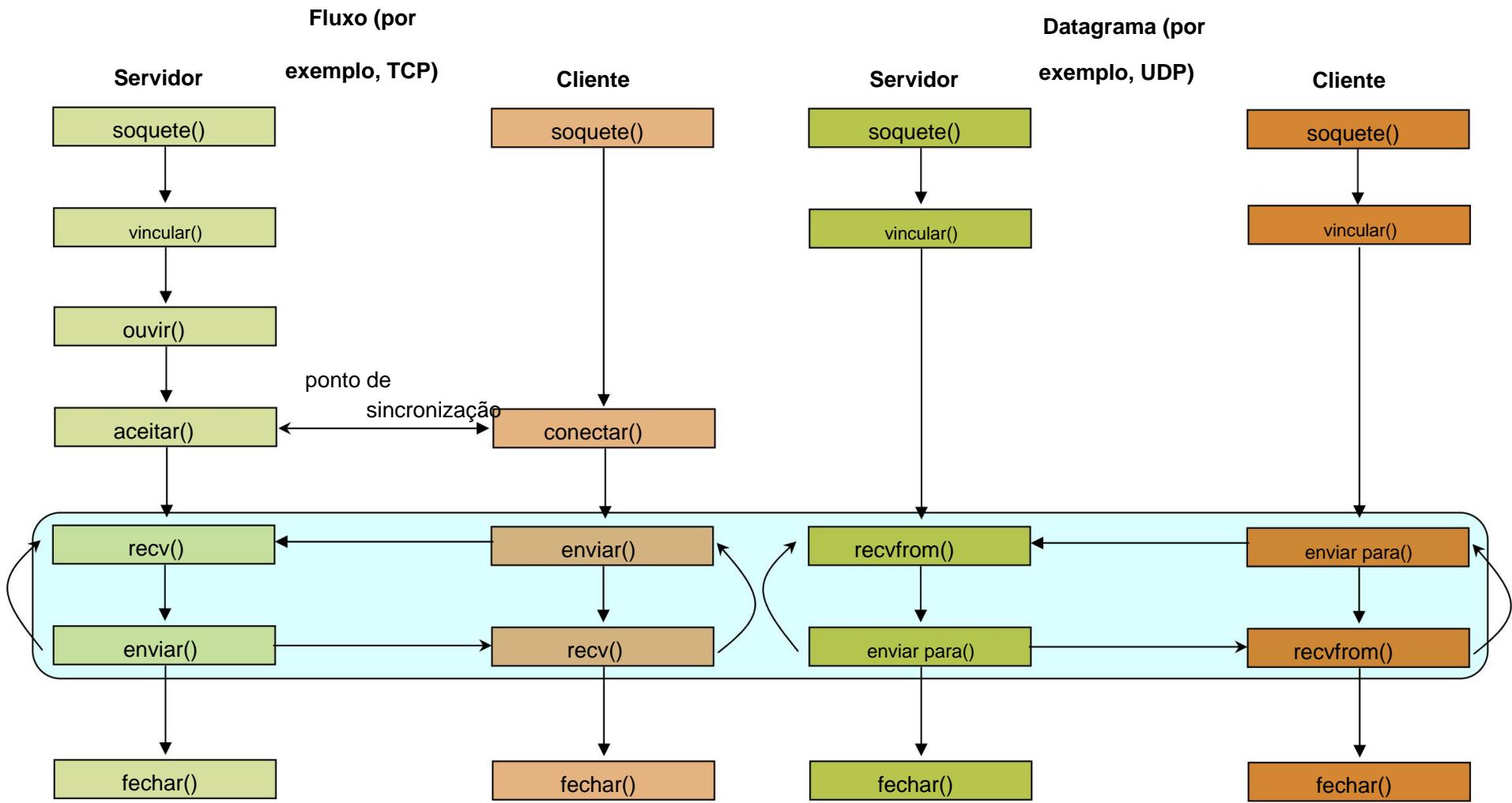
clientAddr: struct sockaddr, endereço do participante ativo
preenchido no retorno

addrLen: sizeof(clientAddr): parâmetro valor/resultado
deve ser definido adequadamente antes da chamada
ajustado no retorno

aceitar()

está bloqueando: aguarda a conexão antes de retornar
desenfileira a próxima conexão na fila para o soquete (sockid)

Comunicação Cliente-Servidor - Unix



Trocando dados com o stream socket

int count = send(sockid, msg, msgLen, flags); msg: const void[], mensagem a

ser transmitida msgLen: inteiro, comprimento da
mensagem (em bytes) a ser transmitida flags: inteiro, opções
especiais, geralmente apenas 0 count: # bytes
transmitidos (-1 se erro)

int count = recv(sockid, recvBuf, bufLen, flags); recvBuf: void[], armazena bytes

recebidos bufLen: # bytes recebidos flags:
inteiro, opções especiais,
geralmente apenas 0 count: # bytes recebidos (-1
se erro)

As chamadas estão bloqueando

retorna somente após os dados serem enviados/recebidos

Trocando dados com soquete de datagrama

```
int count = sendto(sockid, msg, msgLen, flags, &foreignAddr, addrlen);
```

msg, msgLen, flags, count: o mesmo com send()

foreignAddr: struct sockaddr, endereço do destino

addrLen: tamanho de (endereço estrangeiro)

```
int count = recvfrom(sockid, recvBuf, bufLen, flags, &clientAddr, addrlen);
```

recvBuf, bufLen, flags, count: o mesmo com recv()

clientAddr: struct sockaddr, endereço do cliente

addrLen: tamanho de (clientAddr)

As chamadas estão bloqueando

retorna somente após os dados serem enviados/recebidos

Exemplo - Eco

Um cliente se comunica com um servidor “eco”

O servidor simplesmente ecoa tudo o que recebe de volta para o cliente

Exemplo - Echo usando socket de fluxo

O servidor começa se preparando para receber conexões de clientes...

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```
/* Cria socket para conexões de entrada */  
if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)  
    DieWithError("socket() falhou");
```

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. **Crie um soquete TCP**
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```

de Internet */ echoServAddr.sin_family =          /* Família de endereços
AF_INET; echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Qualquer interface de
Porta local */ echoServAddr.sin_port = htons(echoServPort);    entrada */
/* */

se (bind(servSock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
DieWithError("bind() falhou");

```

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. **Atribuir uma porta ao soquete**
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```
/* Marque o soquete para que ele escute conexões de entrada */ if  
(listen(servSock, MAXPENDING) < 0)  
    DieWithError("listen() falhou");
```

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. **Definir o soquete para escutar**
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```
para (;;) /* Executar para sempre */ {
    cIntLen = tamanho de (echoCIntAddr);

    se ((clientSock=aceitar(servSock,(struct sockaddr *)&echoCIntAddr,&cIntLen))<0)
        DieWithError("falha ao aceitar()");
    ...
}
```

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

O servidor agora está bloqueado aguardando conexão de um cliente

...

Um cliente decide falar com o servidor

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```
/* Crie um soquete de fluxo confiável usando  
TCP */ if ((clientSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)  
    DieWithError("socket() falhou");
```

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```

de Internet */ echoServAddr.sin_family = /* Família de endereços
AF_INET; echoServAddr.sin_addr.s_addr = inet_addr(echoservIP); /* Endereço IP do
servidor*/ echoServAddr.sin_port = htons(echoServPort); /* Porta do servidor */

se (conectar(clientSock, (struct sockaddr *)&echoServAddr, tamanho
de(echoServAddr)) < 0)
    DieWithError("falha na conexão");

```

Cliente

1. Crie um soquete TCP
2. **Estabeleça conexão**
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

O procedimento de aceitação do servidor agora está desbloqueado e retorna o soquete do cliente

```
para (;;) /* Executar para sempre */ {

    cIntLen = tamanho de (echoCIntAddr);

    se ((clientSock=aceitar(servSock,(struct sockaddr *)&echoCIntAddr,&cIntLen))<0)
        DieWithError("falha ao aceitar()");
    ...
}
```

Cliente

1. Crie um soquete TCP
2. **Estabeleça conexão**
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - Aceitar nova conexão**
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```
echoStringLen = strlen(echoString); /* Determina o comprimento da entrada */  
  
/* Envia a string para o servidor */  
if (send(clientSock, echoString, echoStringLen, 0) != echoStringLen)  
    DieWithError("send() enviou um número diferente de bytes do que o esperado");
```

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

```

/* Receber mensagem do cliente
 */ if ((recvMsgSize = recv(cIntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
    DieWithError("recv() falhou"); /* Envia a string
recebida e recebe novamente até o fim da transmissão */ while (recvMsgSize
> 0) { /* zero indica o fim da transmissão */ if (send(clientSocket, echobuffer,
recvMsgSize, 0) != recvMsgSize)
    DieWithError("send() falhou"); se
((recvMsgSize = recv(clientSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
    DieWithError("recv() falhou");
}

```

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
- 3. Comunique-se**
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete **3.**
Definir o soquete para escutar
- 4.** Repetidamente: **a.**
Aceitar nova conexão **b. Comunicar**
- c. Feche a conexão**

Exemplo - Echo usando socket de fluxo

Da mesma forma, o cliente recebe os dados do servidor

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. **Comunique-se**
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. **Comunicar**
 - c. Feche a conexão

Exemplo - Echo usando socket de fluxo

`fechar(clientSock);`

`fechar(clientSock);`

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
- 4. Feche a conexão**

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão**

Exemplo - Echo usando socket de fluxo

O servidor agora está bloqueado aguardando conexão de um cliente

...

Cliente

1. Crie um soquete TCP
2. Estabeleça conexão
3. Comunique-se
4. Feche a conexão

Servidor

1. Crie um soquete TCP
2. Atribuir uma porta ao soquete
3. Definir o soquete para escutar
4. Repetidamente:
 - a. Aceitar nova conexão
 - b. Comunicar
 - c. Feche a conexão

Exemplo - Echo usando socket de datagrama

```
/* Cria socket para enviar/receber datagramas */ if  
((servSock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)  
DieWithError("socket() falhou");
```

```
/* Cria um datagrama/soquete UDP  
*/ if ((clientSock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)  
DieWithError("socket() falhou");
```

Cliente

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete 3.
- Comunicar
4. Feche o soquete

Servidor

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete 3.
- Comunicar
- repetidamente

Exemplo - Echo usando socket de datagrama

```

echoServAddr.sin_family = AF_INET;                                /* Família de endereços de internet */
echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Qualquer interface de entrada */
echoServAddr.sin_port = htons(echoServPort);                      /* Porta local */

se (bind(servSock, (struct sockaddr *)&echoServAddr, sizeof(echoServAddr)) < 0)
    DieWithError("bind() falhou");

```

```

echoClientAddr.sin_family = AF_INET;                                /* Família de endereços de internet */
echoClientAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Qualquer interface de entrada */
echoClientAddr.sin_port = htons(echoClientPort);                      /* Porta local */

se(vincular(clientSock,(struct sockaddr *)&echoClientAddr,tamanhode(echoClientAddr))<0)
    DieWithError("falha na conexão()");

```

Cliente

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete
3. Comunique-se
4. Feche o soquete

Servidor

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete
3. Repetidamente
Comunicar

Exemplo - Echo usando socket de datagrama

```

echoServAddr.sin_family = AF_INET;                                /* Família de endereços de internet */
echoServAddr.sin_addr.s_addr = inet_addr(echoservIP); /* Endereço IP do servidor*/
echoServAddr.sin_port = htons(echoServPort);                      /* Porta do servidor */

echoStringLen = strlen(echoString); /* Determina o comprimento da entrada */

/* Envia a string para o servidor */
se (enviar para( clientSock, echoString, echoStringLen, 0, (struct sockaddr *)
    &echoServAddr, sizeof(echoServAddr)) != echoStringLen)

DieWithError("send() enviou um número diferente de bytes do que o esperado");

```

Cliente

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete
3. Comunique-se
4. Feche o soquete

Servidor

1. Crie um soquete UDP
2. **Atribuir uma porta ao soquete**
3. Repetidamente
Comunicar

Exemplo - Echo usando socket de datagrama

```

para (;;) /* Executar para sempre
*{
    clientAddrLen = sizeof(echoClientAddr) /* Define o tamanho do parâmetro in-out */
    /*Bloquear até receber mensagem do cliente*/ if
    ((recvMsgSize = recvfrom(servSock, echoBuffer, ECHOMAX, 0),
        (struct sockaddr *) &echoClientAddr, tamanho de(echoClientAddr)) < 0)
        DieWithError("recvfrom() falhou");

    se (sendto(servSock, echobuffer, recvMsgSize, 0,
        (struct sockaddr *) &echoClientAddr, tamanho de (echoClientAddr)) != recvMsgSize)
        DieWithError("send() falhou");
}

```

Cliente

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete 3.
- Comunicar**
4. Feche o soquete

Servidor

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete 3.
- Comunicar**
- repetidamente

Exemplo - Echo usando socket de datagrama

Da mesma forma, o cliente recebe os dados do servidor

Cliente

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete **3.**
- Comunicar**
4. Feche o soquete

Servidor

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete **3.**
- Comunicar**
- repetidamente

Exemplo - Echo usando socket de datagrama

```
fechar(clientSock);
```

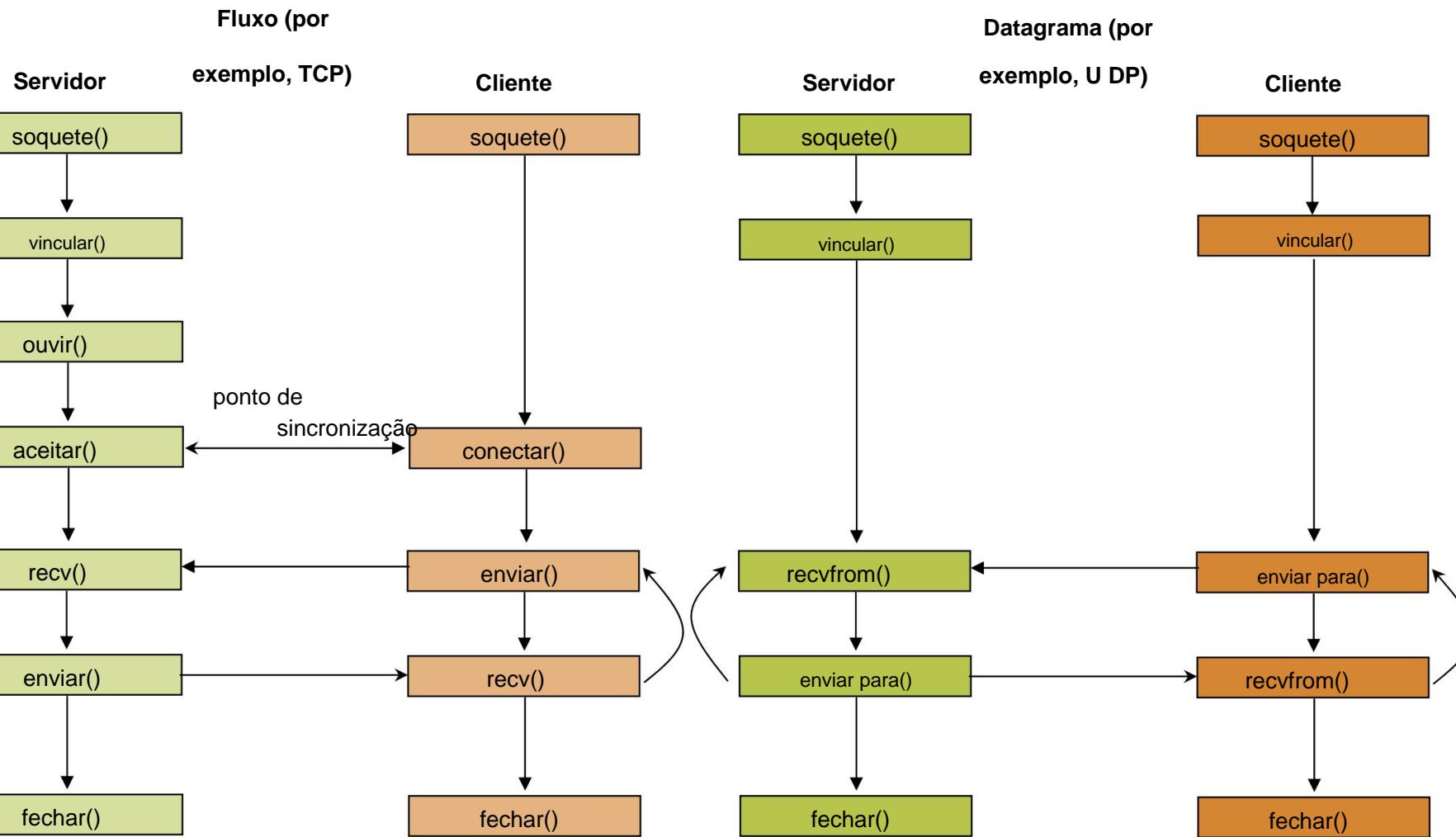
Cliente

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete
3. Comunique-se
4. Feche o soquete

Servidor

1. Crie um soquete UDP
2. Atribuir uma porta ao soquete
3. Repetidamente
Comunicar

Comunicação Cliente-Servidor - Unix



Construindo Mensagens - Codificando Dados

O cliente deseja enviar dois inteiros x e y para o servidor

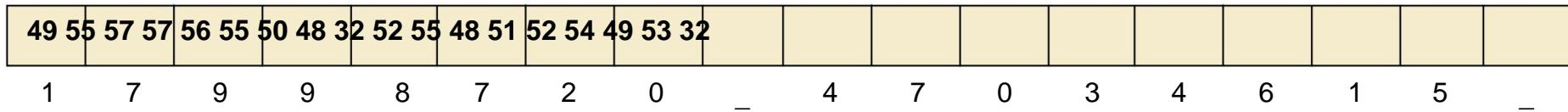
1^a Solução: Codificação de Caracteres

por exemplo ASCII

a mesma representação é usada para imprimi-los ou exibi-los na tela

permite o envio de números arbitrariamente grandes (pelo menos em princípio)

por exemplo $x = 17.998.720$ e $y = 47.034.615$



```
sprintf(msgBuffer, "%d %d ", x, y);
enviar(clientSocket, strlen(msgBuffer), 0);
```

Construindo Mensagens - Codificando Dados

Armadilhas

o segundo delimitador é necessário

caso contrário o servidor não será capaz de separá-lo do que quer que seja que o siga

msgBuffer deve ser grande o suficiente

strlen conta apenas os bytes da mensagem

não o nulo no final da string

Esta solução não é eficiente

cada dígito pode ser representado usando 4 bits, em vez de um byte

é inconveniente manipular números

2^a Solução: Enviando os valores de x e y

Construindo Mensagens - Codificando Dados

2^a Solução: Envio dos valores de x e y armadilha: formato nativo

inteiro um protocolo é usado quantos

bits são usados para

cada inteiro que tipo de codificação é usada (por

exemplo, complemento de dois, sinal/magnitude, sem sinal)

1^a Implementação

```
typedef struct { int
    x,y; }
estrutura de mensagem;
...
msgStruct.x = x; msgStruct.y = y; enviar(clientSock,
&msgStruct, tamanho(msgStruct), 0);
```

2^a Implementação

```
enviar(clientSock, &x, tamanho de(x)), 0);
enviar(clientSock, &y, tamanho de(y)), 0);
```

A segunda implementação
funciona em qualquer caso?

Construindo Mensagens - Ordenação de Bytes

Endereço e porta são armazenados como

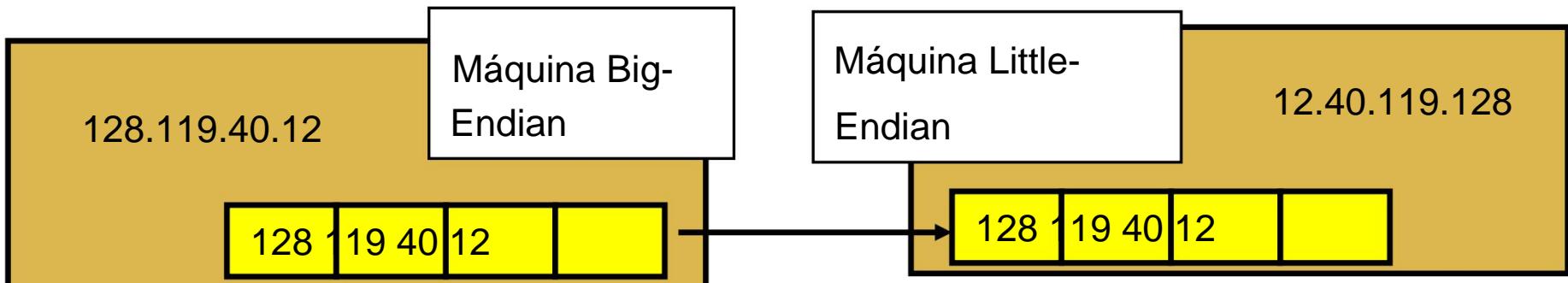
```
inteiros u_short sin_port; (16  
bits) in_addr sin_addr; (32 bits)
```

Problema:

diferentes máquinas/SOs usam diferentes ordens de palavras

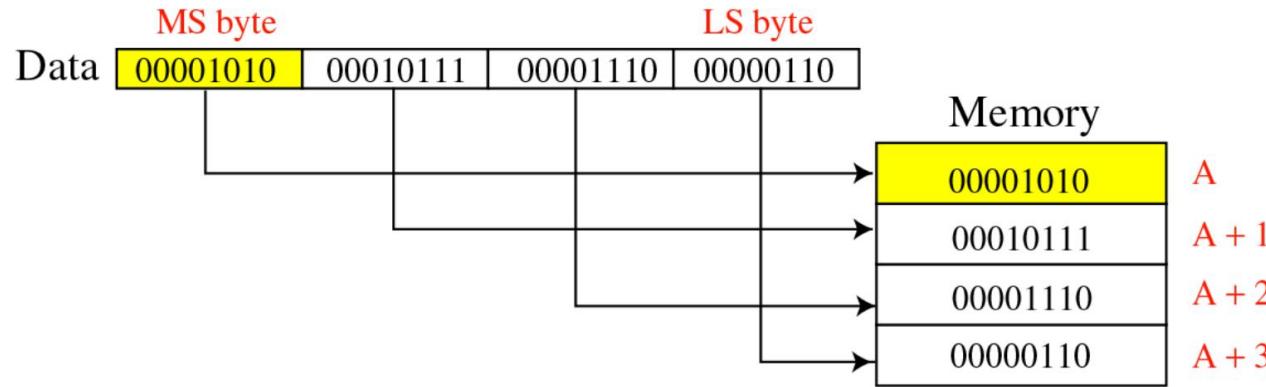
- little-endian: bytes inferiores
- primeiro
- big-endian: bytes

superiores primeiro essas máquinas podem se comunicar entre si pela rede

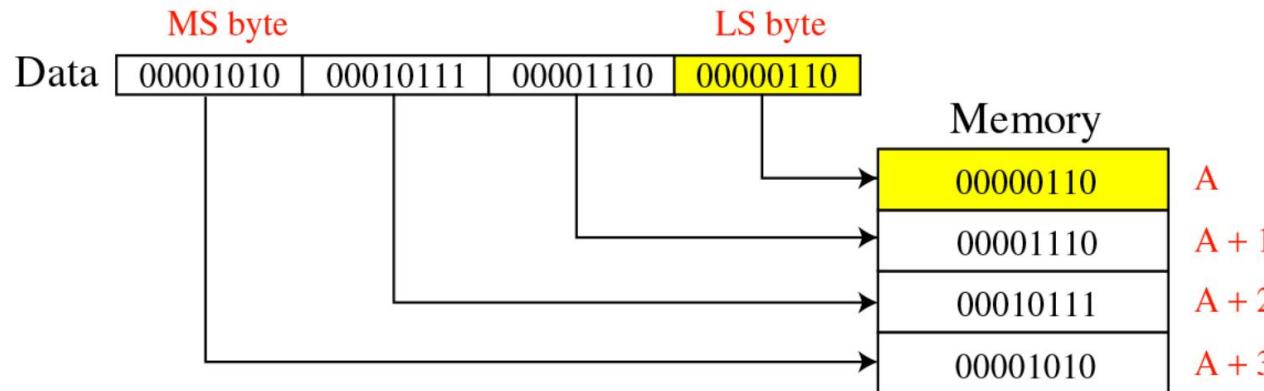


Construindo Mensagens - Ordenação de Bytes

BigEndian:



Pequena Endiana:



Construindo Mensagens - Ordenação de Bytes -

Solução: Ordenação de bytes de rede

Ordenação de bytes do host: a ordenação de bytes usada por um host (grande ou pequeno)

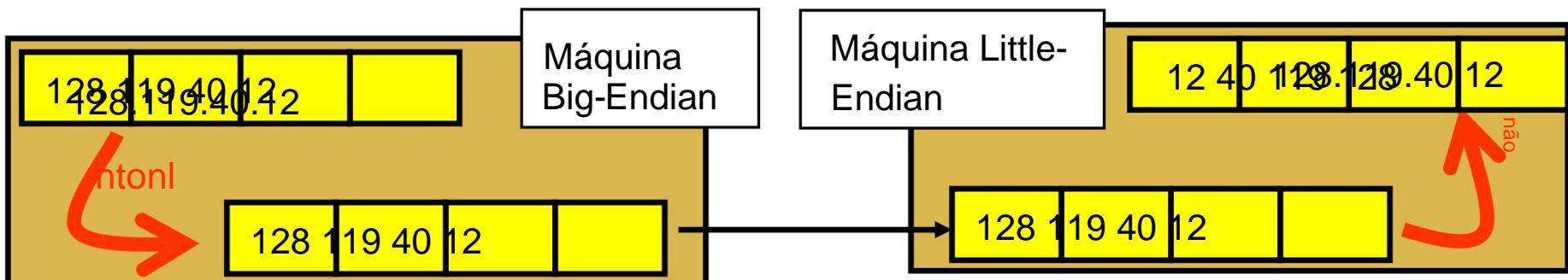
Network Byte-Ordering: a ordenação de bytes usada pela rede – sempre big-endian

```
u_long htonl(u_long x); u_curto  
htons(u_curto x);
```

```
u_longo ntohs(u_longo x);  
u_curto ntohs(u_curto x);
```

Em máquinas big-endian, essas rotinas não fazem nada

Em máquinas little-endian, eles invertem a ordem dos bytes



Construindo Mensagens - Ordenação de Bytes -

Cliente

Exemplo

```
unsigned short clientPort, mensagem; unsigned int messageLength;

servPort = 1111;
mensagem = htons(clientPort);
messageLength = sizeof(mensagem);

se (sendto( clientSock, mensagem, messageLength, 0, (struct sockaddr *)
&echoServAddr, sizeof(echoServAddr)) != messageLength)

    DieWithError("send() enviou um número diferente de bytes do que o esperado");
```

Servidor

```
clientPort curto não assinado, recvBuffer; int não
assinado recvMsgSize;

se ( recvfrom(servSock, &recvBuffer, sizeof(unsigned int), 0),
(struct sockaddr *) &echoClientAddr, tamanho de(echoClientAddr)) < 0)
    DieWithError("recvfrom() falhou");

clientPort = ntohs(recvBuffer); printf ("Porta
do cliente: %d", clientPort);
```

Construindo Mensagens - Alinhamento e Preenchimento

considere a seguinte estrutura de 12 bytes

```
typedef estrutura {
    int x;
    curto x2;
    int e;
    curto y2;
} Estrutura de mensagem;
```

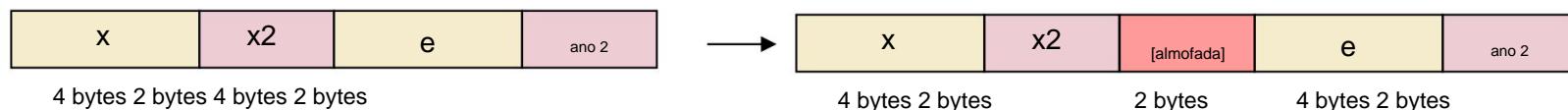
Após a compilação, será uma estrutura de 14 bytes !

Por quê? **Alinhamento!**

Lembre-se das seguintes regras:

as estruturas de dados são alinhadas ao máximo, de acordo com o tamanho do maior inteiro nativo

outros campos multibyte são alinhados ao seu tamanho, por exemplo, o endereço de um inteiro de quatro bytes será divisível por quatro



Isso pode ser evitado

incluir preenchimento na estrutura de dados
reordenar campos

```
typedef estrutura {
    int x;
    curto x2;
    bloco de caracteres[2];
    int e;
    curto y2;
} Estrutura de mensagem;
```

```
typedef estrutura {
    int x;
    int e;
    curto x2;
    curto y2;
} Estrutura de mensagem;
```

Construindo mensagens - Enquadramento e análise sintática

O enquadramento é o problema de formatar as informações para que o receptor possa analisar as mensagens

Analizar significa localizar o início e o fim da mensagem

Isso é fácil se os campos tiverem tamanhos fixos

por exemplo, msgStruct

Para representações de strings de texto é mais difícil

Solução: uso de delimitadores apropriados

é necessário cuidado, pois uma chamada de recv pode retornar as mensagens enviadas por várias chamadas de send

Opções de soquete

getsockopt e setsockopt permitem que valores de opções de soquete sejam consultados e definidos, respectivamente

int getsockopt (sockid, nível, optName, optVal, optLen);

sockid: inteiro, descriptor de soquete

nível: inteiro, as camadas da pilha de protocolos (socket, TCP, IP)

optName: inteiro, opção

optVal: ponteiro para um buffer; ao retornar, contém o valor da opção especificada

optLen: inteiro, parâmetro de entrada e saída

retorna -1 se ocorreu um erro

int setsockopt (sockid, nível, optName, optVal, optLen);

optLen agora é apenas um parâmetro de entrada

Opções de soquete

- Mesa

<i>optName</i>	Type	Values	Description
SOL_SOCKET Level			
SO_BROADCAST	int	0,1	Broadcast allowed
SO_KEEPALIVE	int	0,1	Keepalive messages enabled (if implemented by the protocol)
SO_LINGER	linger{}	time	Time to delay close() return waiting for confirmation (see Section 6.4.2)
SO_RCVBUF	int	bytes	Bytes in the socket receive buffer (see code on page 44 and Section 6.1)
SO_RCVLOWAT	int	bytes	Minimum number of available bytes that will cause recv() to return
SO_REUSEADDR	int	0,1	Binding allowed (under certain conditions) to an address or port already in use (see Section 6.4 and 6.5)
SO_SNDDLOWAT	int	bytes	Minimum bytes to send a packet
SO_SNDBUF	int	bytes	Bytes in the socket send buffer (see Section 6.1)
IPPROTO_TCP Level			
TCP_MAX	int	seconds	Seconds between keepalive messages.
TCP_NODELAY	int	0,1	Disallow delay for data merging (Nagle's algorithm)
IPPROTO_IP Level			
IP_TTL	int	0-255	Time-to-live for unicast IP packets
IP_MULTICAST_TTL	unsigned char	0-255	Time-to-live for multicast IP packets (see MulticastSender.c on page 81)
IP_MULTICAST_LOOP	int	0,1	Enables multicast socket to receive packets it sent
IP_ADD_MEMBERSHIP	ip_mreq{}	group address	Enables reception of packets addressed to the specified multicast group (see MulticastReceiver.c on page 83)—set only
IP_DROP_MEMBERSHIP	ip_mreq{}	group address	Disables reception of packets addressed to the specified multicast group—set only

Opções de Socket - Exemplo

Buscar e então duplicar o número atual de bytes no buffer de recebimento do soquete

```
int tamanho do buffer rcv;
int tamanho da meia;
...
/* Recupera e imprime o tamanho do buffer
padrão */ sockOptSize = sizeof(recvBuffSize); if
(getsockopt(sock, SOL_SOCKET, SO_RCVBUF, &recvBufferSize, &sockOptSize) < 0)
    DieWithError("getsockopt() falhou");
printf("Tamanho inicial do buffer de recebimento: %d\n", recvBufferSize);

/* Duplique o tamanho do buffer
*/ recvBufferSize *= 2;

/* Defina o tamanho do buffer para o
novo valor */ if (setsockopt(sock, SOL_SOCKET, SO_RCVBUF, &recvBufferSize,
        sizeof(recvBufferSize)) < 0)
    DieWithError("getsockopt() falhou");
```

Lidando com bloqueio de chamadas

Muitas das funções que vimos bloqueiam (por padrão) até um certo evento

aceitar: até que uma conexão seja

estabelecida conectar: até que a conexão seja estabelecida

recv, recvfrom: até que um pacote (de dados) seja recebido e se

um pacote for perdido (no soquete do datagrama)?

send: até que os dados sejam enviados para o buffer do

soquete sendto: até que os dados sejam fornecidos ao subsistema de rede

Para **programas simples**, o bloqueio é conveniente. E quanto aos

programas mais complexos ?

múltiplas conexões

simultâneas envia e recebe

realizando simultaneamente processamento não relacionado à rede

Lidando com bloqueio de chamadas

Soquetes não bloqueantes

E/S assíncrona

Tempos limite

Soquetes não bloqueantes

Se uma operação puder ser concluída imediatamente, o sucesso será retornado; caso contrário, uma falha será retornada (geralmente -1)

errno está definido corretamente, para distinguir esta falha (de bloqueio) de outras
- (EINPROGRESS para conectar, EWOULDBLOCK para o outro)

1^a Solução: **int fcntl (sockid, comando, argumento);**

sockid: inteiro, descritor de soquete

comando: inteiro, a operação a ser executada (**F_GETFL, F_SETFL**)

argumento: longo, por exemplo **O_NONBLOCK**

fcntl (sockid, F_SETFL, O_NONBLOCK);

2^a Solução: parâmetro flags de send, recv, sendto, recvfrom

MSG_NÃO_ESPERE

não suportado por todas as implementações

Sinais

Fornece um mecanismo para o sistema operacional notificar os processos de que certos eventos ocorrem

por exemplo, o usuário digitou o caractere “interrupção” ou um temporizador expirou, os sinais são entregues **de forma assíncrona** após a entrega do sinal ao programa

pode ser **ignorado**, o processo nunca está ciente disso o programa

é **encerrado à força** pelo sistema operacional uma **rotina de**

tratamento de sinal, especificada pelo programa, é executada

isso acontece em um thread diferente o

sinal é **bloqueado**, até que o programa tome uma ação para permitir sua entrega cada processo (ou thread) tem uma **máscara** correspondente Cada sinal

tem um **comportamento padrão**, por exemplo

SIGINT (ou seja, Ctrl+C) causa o término ele pode ser

alterado usando sigaction()

Os sinais podem ser **aninhados** (ou seja, enquanto um está sendo manipulado, outro é entregue)

Sinais

int sigaction(whichSignal, &newAction, &oldAction); whichSignal: inteiro newAction:

struct sigaction, define o

novo comportamento oldAction: struct sigaction, se não for NULL, então

o comportamento anterior é copiado, retorna 0 em caso de sucesso, -1 caso contrário

```
struct sigaction { void
    (*sa_handler)(int); /* Manipulador de sinal */ sigset_t
    sa_mask; int sa_flags;
};
```

/* Sinais a serem bloqueados durante a execução do manipulador */
 /* Sinalizadores para modificar o comportamento padrão */

sa_handler determina qual das três primeiras possibilidades ocorre quando o sinal é entregue, ou seja, não é mascarado SIG_IGN,

SIG_DFL, endereço de uma função

sa_mask especifica os sinais a serem bloqueados durante o tratamento de whichSignal

whichSignal é sempre bloqueado, é

implementado como um conjunto de sinalizadores booleanos

```
int sigemptyset (sigset_t *set); /* desativa todos os
sinalizadores */ int sigfullset (sigset_t *set); /* desativa
todos os sinalizadores */ int sigaddset(sigset_t *set, int whichSignal); /* desativa
sinalizadores individuais */ int sigdelset(sigset_t *set, int whichSignal); /* desativa sinalizadores individuais */
```

Sinais - Exemplo

```
#incluir <stdio.h>
#incluir <sinal.h>
#include <unistd.h>

vazio DieWithError(char *errorMessage);
void InterruptSignalHandler(int TipoDeSinal);

int principal (int argc, char *argv[]) {
    manipulador de struct sigaction ; /* Estrutura de especificação do manipulador de sinal */
    manipulador.sa_handler = InterruptSignalHandler; /* Define a função do manipulador */
    if (sigfillset(&handler.sa_mask) < 0) /* Cria uma máscara que mascara todos os sinais */
        DieWithError ("sigfillset() falhou");
    manipulador.sa_flags = 0;
    se (sigaction(SIGINT, &handler, 0) < 0) /* Define o tratamento de sinais para sinais de interrupção */
        DieWithError ("sigaction() falhou");
    para(;;) pausa(); sair(0); /* Suspender programa até que o sinal seja recebido */
}

void InterruptHandler (int tipo de sinal) {
    printf ("Interrupção recebida. Saindo do programa.\n");
    saída(1);
}
```

E/S assíncrona

Soquetes não bloqueantes requerem “polling”

Com E/S assíncronas, o sistema operacional informa o programa quando uma chamada de soquete é concluída

o sinal **SIGIO** é entregue ao processo, quando algum evento relacionado a E/S ocorre no soquete

Três etapas: /*

i. informar o sistema sobre a disposição desejada do sinal */

manipulador de struct sigaction;

manipulador.sa_handler=SIGIOHandler; if

(sigfillset(&handler.sa_mask) < 0) DieWithError("..."); manipulador.sa_flags = 0;

if (sigaction(SIGIO, &handler,

0) < 0) DieWithError("...");

/* ii. garantir que os sinais relacionados ao soquete serão entregues a este processo */

se (fcntl(meia, F_SETOWN, getpid()) < 0) DieWithError();

/* iii. marcar o soquete como preparado para E/S assíncrona */

se (fcntl(meia, F_SETFL, O_NONBLOCK | FASYNC) < 0) DieWithError()

Tempos limite

Usando E/S assíncronas, o sistema operacional informa o programa sobre a ocorrência de um evento relacionado a E/S

o que acontece se um pacote UPD for perdido?

Podemos precisar saber se algo não acontece depois de algum tempo **unsigned int alarm (unsigned int secs);**

inicia um cronômetro que expira após o número especificado de segundos (**secs**)
retorna

o número de segundos restantes até que qualquer alarme previamente agendado seja emitido, ou zero se

não houver nenhum alarme previamente agendado

o processo recebe o sinal **SIGALARM** quando o temporizador expira e o errno é definido como **EINTR**

E/S assíncrona - Exemplo

```
/* Informa o sistema sobre a disposição desejada do sinal */ struct  
sigaction myAction; myAction.sa_handler  
= CatchAlarm; if (sigfillset(&myAction.sa_mask)  
< 0) DieWithError("..."); myAction.sa_flags = 0; if (sigaction(SIGALARM, &handler,  
0) < 0) DieWithError("...");  
  
/* Definir alarme */  
  
alarme(TIMEOUT_SECS);  
  
/* Bloqueio de chamada recebido */  
se (recvfrom(meia, echoBuffer, ECHOMAX, 0, ... ) < 0) {  
    if (errno = EINTR) ... /* Alarme disparou */  
    senão DieWithError("recvfrom() falhou");  
}  
}
```

Servidor de soquete de fluxo iterativo

Atende um cliente por vez

Cientes adicionais podem se conectar enquanto um está sendo atendido
conexões são estabelecidas

eles podem enviar solicitações

mas o servidor responderá após terminar com o primeiro cliente

Funciona bem se cada cliente exigir uma quantidade pequena e limitada de
trabalho do servidor

caso contrário, os clientes sofrerão longos atrasos

Servidor Iterativo - Exemplo: echo usando stream socket

```
#include <stdio.h> /* para printf() e fprintf() */
#include <sys/socket.h> /* para socket(), bind(), connect(), recv() e send() */
#include <arpa/inet.h> /* para sockaddr_in e inet_ntoa() */
#include <stdlib.h> /* para atoi() e exit() */
#include <string.h> /* para memset() */
#include <unistd.h> /* para fechar() */

#define MAXPENDENTE 5      /* Máximo de solicitações de conexão pendentes */

void DieWithError(char *errorMessage); /* Função de tratamento de erros */
void HandleTCPClient(int clntSocket); /* Função de manipulação de cliente TCP */

int principal(int argc, char *argv[]) {
    int servSock; int                         /* Descritor de soquete para servidor */
    clntSock; struct                          /* Descritor de soquete para cliente */

    sockaddr_in echoServAddr; /* Endereço local */
    struct sockaddr_in echoClntAddr; /* Endereço do cliente */
    echoServPort curto não assinado; int clntLen      /* Porta do servidor */
    não assinado;                                /* Comprimento da estrutura de dados do endereço do cliente */

    se (argc != 2)                            /* Teste para número correto de argumentos */
        { fprintf(stderr, "Uso: %s <Porta do Servidor>\n", argv[0]);
          saída(1);
    }

    echoServPort = atoi(argv[1]); /* Primeiro argumento: porta local */

    /* Cria socket para conexões de entrada */
    se ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() falhou");

    ...
}
```

Servidor Iterativo - Exemplo: echo usando stream socket

```

...
/* Construir estrutura de endereço local */ memset(&echoServAddr,
0, sizeof(echoServAddr)); /* Estrutura de saída zero */ echoServAddr.sin_family = AF_INET; echoServAddr.sin_addr.s_addr
= htonl(INADDR_ANY); /* Qualquer interface de entrada */ /* Família de endereços de internet */
/* echoServAddr.sin_port = htons(echoServPort); /* Porta local */

/* Vincular ao endereço local */ if (bind(servSock,
(struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
    DieWithError("bind() falhou");

/* Marque o soquete para que ele escute conexões de entrada */ if (listen(servSock, MAXPENDING) <
0)
    DieWithError("listen() falhou");

para (;;) /* Executar para sempre */ {

    /* Define o tamanho do parâmetro in-out */ clntLen =
    sizeof(echoCIntAddr);

    /* Aguarde a conexão de um cliente */ if (((cIntSock =
accept(servSock, (struct sockaddr *) &echoCIntAddr, &clntLen)) < 0)

        DieWithError("falha ao aceitar()");

    /* cIntSock está conectado a um cliente! */

    printf("Tratando cliente %s\n", inet_ntoa(echoCIntAddr.sin_addr));

    HandleTCPClient(cIntSock);
}

/* NÃO ALCANÇADO */
}

```

Servidor Iterativo - Exemplo: echo usando stream socket

```
#define RCVBUFSIZE 32          /* Tamanho do buffer de recebimento */

void HandleTCPClient(int cIntSocket)
{
    char echoBuffer[RCVBUFSIZE]; int recvMsgSize;           /* Buffer para sequência de eco */
                                                               /* Tamanho da mensagem recebida */

    /* Receber mensagem do cliente */
    se ((recvMsgSize = recv(cIntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
        DieWithError("recv() falhou");

    /* Envia a string recebida e recebe novamente até o final da transmissão */
    enquanto (recvMsgSize > 0) {                           /* zero indica fim da transmissão */

        /* Ecoar mensagem de volta para o cliente */
        se (enviar(cIntSocket, echoBuffer, recvMsgSize, 0) != recvMsgSize)
            DieWithError("send() falhou");

        /* Verifique se há mais dados para receber */
        se ((recvMsgSize = recv(cIntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
            DieWithError("recv() falhou");
    }

    fechar(cIntSocket);          /* Fechar socket do cliente */
}
```

Multitarefa - Processo por cliente

Para cada solicitação de conexão do cliente, um novo processo é criado para lidar com a comunicação

int garfo();

um novo processo é criado, idêntico ao processo de chamada, exceto pelo seu ID de processo e pelo valor de retorno que ele recebe de fork()

retorna 0 para o processo **filho** e o ID do processo do novo filho para **o pai**

Cuidado:

quando um processo filho termina, ele não desaparece automaticamente
use waitpid() para pai para “colher” zumbis

Multitarefa - Processo por cliente

- Exemplo: eco usando socket de fluxo

```
#incluir <sys/wait.h>                                /* para waitpid() */

int principal(int argc, char *argv[]) {
    int servSock; int                                /* Descritor de soquete para servidor */
    cIntSock; unsigned                               /* Descritor de soquete para cliente */
    short echoServPort; pid_t processID; unsigned   /* Porta do servidor */
    int childProcCount = 0; /* ID do processo de fork()*/
    Número de processos filhos */

    se (argc != 2) { /* Teste para número correto de argumentos */
        { fprintf(stderr, "Uso: %s <Porta do Servidor>\n", argv[0]);
        saída(1);
    }
    echoServPort = atoi(argv[1]); /* Primeiro argumento: porta local */

    servSock = CreateTCPServerSocket(echoServPort);

    para (;;) { /* Executar para sempre */
        cIntSock = AceitarTCPConnection(servSock);

        if ((processID = fork()) < 0) DieWithError ("fork() falhou"); /* Bifurcar processo filho */
        senão se (processID = 0) { /* Este é o processo filho */
            { fechar(servSock); /* criança fecha soquete de escuta */
            HandleTCPClient(cIntSock); saída(0);
            /* processo filho termina */
        }

        fechar(cIntSock); /* pai fecha soquete filho */
        filhoProcCount++; /* Incrementa o número de processos filho pendentes */
    }
}
```

Multitarefa - Processo por cliente

- Exemplo: eco usando socket de fluxo

```
...
enquanto (childProcCount) {
    processID = waitpid((pid_t)-1, NULL, WHOANG); se (processID < 0)
        DieWithError ("...");
    senão se (processID == 0) quebrar; senão
        childProcCount--;
    }
}
/* NÃO ALCANÇADO */
```

```
/* Limpe todos os zumbis */
/* Espera sem bloqueio */
/* Nenhum zumbi para esperar */
/* Limpou a sujeira de uma criança */
```

Multitarefa - Thread por cliente

Bifurcar um novo processo é caro

duplicar todo o estado (memória, pilha, descritores de arquivo/socket, ...)

Os threads diminuem esse custo ao permitir multitarefa dentro do mesmo processo

threads compartilham o mesmo espaço de endereço (código e dados)

Um exemplo é fornecido usando POSIX Threads

Multitarefa - Thread por cliente

- Exemplo: echo usando socket de fluxo

```
#incluir <pthread.h>                                /* para threads POSIX */

vazio *ThreadMain(void *arg)                         /* Programa principal de uma thread */

estrutura ThreadArgs { int                           /* Estrutura de argumentos a serem passados para o thread do cliente */
    clntSock;                                       /* descriptor de soquete para cliente */
};

int principal(int argc, char *argv[]) {
    int servSock, int                               /* Descritor de soquete para servidor */
    clntSock;                                       /* Descritor de soquete para cliente */
    echoServPort curto sem sinal; pthread_t         /* Porta do servidor */
threadID; struct ThreadArgs                      /* ID do thread de pthread_create()*/
*threadArgs;                                       /* Ponteiro para estrutura de argumento para thread */

    se (argc != 2) { fprintf(stderr,      /* Teste para número correto de argumentos */
        "Uso: %s <Porta do Servidor>\n", argv[0]);
        saída(1);
    }
    echoServPort = atoi(argv[1]);                   /* Primeiro argumento: porta local */

    servSock = CreateTCPServerSocket(echoServPort);

    para (;;) { /* Executar para sempre */
        clntSock = AceitarTCPConnection(servSock);

        /* Cria memória separada para argumento do cliente */
        se ((threadArgs = (struct ThreadArgs *) malloc(sizeof(struct ThreadArgs)))) == NULL) DieWithError("...");
        threadArgs -> clntSock = clntSock;

        /* Criar thread do cliente */
        se (pthread_create (&threadID, NULL, ThreadMain, (void *) threadArgs) != 0) DieWithError("...");
    }
    /* NÃO ALCANÇADO */
}
```

Multitarefa - Thread por cliente

- Exemplo: eco usando socket de fluxo

```
vazio *ThreadMain(void *threadArgs)
{
    int cIntMeia;                                /* Descritor de soquete para conexão do cliente */

    pthread_detach(pthread_self()); /* Garante que os recursos do thread sejam desalocados no retorno */

    /* Extraí o descritor do arquivo de socket do argumento */
    cIntSock = ((struct ThreadArgs *) threadArgs) -> cIntSock;
    livre(threadArgs);                           /* Desalocar memória para argumento */

    HandleTCPClient(cIntSock);

    retornar (NULO);
}
```

Multitarefa - Restrita

Tanto o processo quanto o thread incorrem **em sobrecarga**

criação, agendamento e troca de contexto

À medida que seus números aumentam

essa sobrecarga aumenta

depois de algum tempo seria melhor se um cliente fosse bloqueado

Solução: **Multitarefa restrita**. O servidor: começa, criando, vinculando e

escutando um socket

cria uma série de processos, cada um em loop para sempre e aceita conexões
do mesmo soquete

quando uma conexão é estabelecida

o descriptor de soquete do cliente é retornado para apenas um processo
os outros permanecem bloqueados

Multitarefa - Restrita

- Exemplo: eco usando socket de fluxo

```

void ProcessMain(int servSock);                                /* Programa principal do processo */

int principal(int argc, char *argv[]) {
    int servSock;                                              /* Descritor de soquete para servidor*/
    echoServPort curto sem sinal; pid_t                         /* Porta do servidor */
    processID; int                                               /* ID do processo */
    processLimit sem sinal; int processCt                      /* Número de processos filho a serem criados */
    sem sinal;                                                 /* Contador de processos */

    if (argc != 3) { /* Teste o número correto de argumentos */
        fprintf(stderr,"Uso: %s <PORTA DO SERVIDOR> <LIMITE DE FORK>\n", argv[0]);
        saída(1);
    }

    echoServPort = atoi(argv[1]); /* Primeiro argumento: porta local */
    processLimit = atoi(argv[2]); /* Segundo argumento: número de processos filhos */

    servSock = CreateTCPServerSocket(echoServPort);

    para (processCt=0; processCt < processLimit; processCt++)
        se ((processID = fork()) < 0) DieWithError("fork() falhou"); senão se (processID == 0)
            ProcessMain(servSock);                                     /* Bifurcar processo filho */
            /* Se este for o processo filho */

        exit(0); /* As crianças continuarão */
    }
}

void ProcessMain(int servSock) {
    int clntMeia;                                         /* Descritor de soquete para conexão do cliente */

    para (;;) { /* Executar para sempre */
        clntSock = AcceptTCPConnection(servSock);
        printf("com processo filho: %d\n", (unsigned int) getpid());
        HandleTCPClient(clntSock);
    }
}

```

Multiplexação

Até agora, lidamos com um **único** canal de E/S. Talvez precisemos lidar com **vários** canais de E/S, por exemplo, dando suporte ao serviço de eco em várias portas. Problema: de qual soquete o servidor deve aceitar conexões ou receber mensagens?

pode ser resolvido usando soquetes não bloqueantes
mas requer votação

Solução: `select()`
especifica uma lista de descritores para verificar se há blocos de operações de E/S pendentes até que um dos descritores esteja pronto retorna quais descritores estão prontos

Multiplexação

```
int select (maxDescPlus1, &readDescs, &writeDescs, &exceptionDescs,
&timeout);
```

maxDescsPlus1: inteiro, dica do número máximo de descritores **readDescs**:

fd_set, verificado para disponibilidade de entrada imediata **writeDescs**:

fd_set, verificado para capacidade de gravar dados imediatamente

exceptionDescs: **fd_set**, verificado para exceções pendentes

timeout: struct timeval, por quanto tempo ele bloqueia (NULL para sempre)

- retorna o número total de descritores prontos, -1 em caso de erro - altera as listas de descritores para que apenas as posições correspondentes sejam definidas

```
int FD_ZERO (fd_set *descriptorVector); int FD_CLR
(int descriptor, fd_set *descriptorVector); /* remove todos os descritores do vetor */
(int descriptor, fd_set *descriptorVector); /* remove descritor do vetor */ int FD_SET (int descriptor, fd_set
*descriptorVector); /* adiciona descritor ao vetor */ int FD_ISSET (int descriptor, fd_set
*descriptorVector); /* verifica a associação do vetor */
```

```
struct timeval { time_t
    tv_sec; /* segundos */ time_t tv_usec; /*
    microsegundos */
};
```

Multiplexação - Exemplo: echo usando stream socket

```
#incluir <sys/tempo.h>           /* para struct timeval {} */

int principal(int argc, char *argv[])
{
    int *servSock; int
    maxDescriptor; fd_set
    sockSet; tempo limite
    longo ; struct timeval
    selTimeout; int running = 1; int noPorts;
    int port; portNo curto
    sem sinal;

    /* Descritores de soquete para servidor */
    /* Valor máximo do descriptor de soquete */
    /* Conjunto de descritores de soquete para select() */
    /* Valor de tempo limite fornecido na linha de comando */
    /* Tempo limite para select() */
    /* 1 se o servidor deve estar em execução; 0 caso contrário */
    /* Número de portas especificado na linha de comando */
    /* Variável de loop para portas */
    /* Número da porta atual */

    se (argc < 3)                  /* Teste para número correto de argumentos */
        { fprintf(stderr, "Uso: %s <Tempo limite (segs.)> <Porta 1> ... \n", argv[0]);
        saída(1);
    }

    tempo limite = atol(argv[1]);      /* Primeiro argumento: Timeout */
    noPorts = argc - 2;               /* O número de portas é a contagem de argumentos menos 2 */

    servSock = (int *) malloc(noPorts * sizeof(int)); /* Alocar lista de soquetes para conexões de entrada */
    maxDescriptor = -1;                /* Inicializa maxDescriptor para uso por select() */

    para (porta = 0; porta < noPorts; porta++) {
        portNo = atoi(argv[porta + 2]); servSock[porta]
        = CreateTCPServerSocket(portNo); /* Cria soquete de porta */

        /* Cria lista de portas e sockets para manipular portas */ /* Adiciona porta à
        lista de portas. Pula os dois primeiros argumentos */

        se (servSock[porta] > maxDescriptor) maxDescriptor
            = servSock[porta];
    }
    ...
}
```

Multiplexação - Exemplo: echo usando stream socket

```

printf("Iniciando servidor: Pressione Enter para desligar\n"); while (running)
{
    /* Vetor descriptor de soquete zero e definido para soquetes de servidor */
    /* Isso deve ser redefinido toda vez que select() for chamado */
    FD_ZERO(&conjunto de meias);
    FD_SET(STDIN_FILENO, &sockSet); /* Adicionar teclado ao vetor descriptor */ for (port = 0; port <
    noPorts; port++) FD_SET(servSock[port], &sockSet);

    /* Especificação de tempo limite */ /*
    Isso deve ser redefinido toda vez que select() for chamado */ /* tempo limite
    timeout; /* 0 microssegundos */ (seg.) */ selTimeout.tv_sec =
        selTimeout.tv_usec = 0;

    /* Suspender o programa até que o descriptor esteja pronto ou atinja o tempo limite
    */ if (select(maxDescriptor + 1, &sockSet, NULL, NULL, &selTimeout) == 0)
        printf("Nenhuma solicitação de eco por %d segundos...Servidor ainda ativo\n", timeout); else { if (FD_ISSET(0,
    &sockSet))
        { /* Verificar teclado */ printf("Desligando servidor\n"); getchar(); running =
            0;

    } para (porta = 0; porta < noPorts; porta++)
        se (FD_ISSET(servSock[porta], &sockSet)) {
            printf("Solicitação na porta %d: ", porta);
            HandleTCPClient(AcceptTCPConnection(servSock[porta]));
        }
    }

} para (porta = 0; porta < noPorts; porta++) close(servSock[porta]); /* Fechar soquetes */
free(servSock); exit(0); /* Lista livre de soquetes */

}

```

Vários destinatários

Até agora, todos os sockets lidaram com comunicação **unicast**

ou seja, uma comunicação um-para-um, onde uma cópia (“**uni**”) dos dados é enviada (“**cast**”)

e se quisermos enviar dados para vários destinatários?

1ª Solução: unicast uma cópia dos dados para cada destinatário

ineficiente, por

exemplo, considere que estamos conectados à internet por meio de uma linha de 3 Mbps

um servidor de vídeo envia fluxos de 1 Mbps

então, o servidor pode suportar apenas três clientes simultaneamente

2ª Solução: usar suporte de rede

transmissão, todos os hosts da rede recebem a mensagem

multicast, uma mensagem é enviada para algum subconjunto do host

para IP: somente **soquetes UDP** têm permissão para transmitir e multidifundir

Vários destinatários - Transmissão

Apenas o endereço IP muda

Transmissão **local** : para o endereço 255.255.255.255

envie a mensagem para todos os hosts na mesma rede de transmissão não encaminhada pelos roteadores

Transmissão **dirigida** :

para o identificador de rede 169.125 (ou seja, com máscara de sub-rede 255.255.0.0), o endereço de transmissão direcionado é 169.125.255.255

Nenhum endereço de transmissão em toda a rede está disponível

por que?

Para usar a transmissão, as opções do soquete devem ser alteradas:

int transmissãoPermissão = 1;

**setsockopt(meia, SOL_SOCKET, SO_BROADCAST, (void*)
&broadcastPermission, tamanho de(broadcastPermission));**

Vários destinatários - Multicast

Usando endereços **de classe D**

que variam de 224.0.0.0 a 239.255.255.255, os hosts enviam **solicitações multicast** para endereços específicos, um **grupo multicast** é formado

precisamos definir TTL (tempo de vida), para limitar o número de saltos - usando sockopt()

não há necessidade de alterar as opções do socket

Funções Úteis

int atoi(const char *nptr);

converte a porção inicial da string apontada por nptr para int

int inet_aton(const char *cp, struct in_addr *inp);

converte o endereço do host da Internet cp da notação de números e pontos

IPv4 para o formato binário (na ordem de bytes da rede) e o

armazena na estrutura para a qual inp aponta. Ele

retorna um valor diferente de zero se o endereço for válido e 0 se não for

char *inet_ntoa(struct in_addr em);

converte o endereço do host da Internet, fornecido na ordem de bytes da rede, em
uma string na notação decimal com pontos IPv4

```
typedef uint32_t in_addr_t;  
  
estrutura in_addr {  
    em_addr_t s_addr;  
};
```

Funções Úteis

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

retorna o endereço (IP e porta) do peer conectado ao socket sockfd, no buffer apontado por addr

0 é retornado em caso de sucesso; -1 caso contrário

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

retorna o endereço atual ao qual o soquete sockfd está vinculado, no buffer apontado por addr

0 é retornado em caso de sucesso; -1 caso contrário

Serviço de Nome de Domínio

struct hostent *gethostbyname(const char *nome);

retorna uma estrutura do tipo hostent para o nome do host fornecido nome é um nome de host ou um endereço IPv4 na notação de ponto padrão

por exemplo, **gethostbyname("www.csd.uoc.gr");**

struct hostent *gethostbyaddr(const void *addr, socklen_t len, int tipo);

retorna uma estrutura do tipo hostent para o endereço de host fornecido addr de comprimento len e tipo de endereço type

estrutura hostent {

```
char *h_name; /* nome oficial do host */
char **h_aliases; lista de alias (strings) */
int /* h_addrtype; /* tipo de endereço de host (AF_INET) */
Inteiro /* h_length; /* comprimento de endereço */
char **h_addr_list; /* lista de endereços (binário em ordem de bytes de rede) */
}
```

#define h_addr h_addr_list[0] /* para compatibilidade com versões anteriores */

Serviço de Nome de Domínio

```
struct servent *getservbyname(const char *nome, const char *proto);
```

retorna uma estrutura servent para a entrada do banco de dados que corresponde ao nome do serviço usando o protocolo proto. Se proto for NULL, qualquer protocolo será correspondido.

por exemplo `getservbyname("echo", "tcp")` ;

```
struct servent *getservbyport(int porta, const char *proto);
```

retorna uma estrutura de serviço para a entrada do banco de dados que corresponde ao nome do serviço usando a porta porta

```
estrutura servent {  
    char *s_nome; char /* nome oficial do serviço */  
    **s_aliases; int s_esporte; /* lista de nomes alternativos (strings) */  
    char *s_proto; /* número da porta do serviço */  
    /* protocolo a ser usado ("tcp" ou "udp") */  
}
```

Compilando e executando

incluir os arquivos de cabeçalho necessários

Exemplo:

```
milo:~/CS556/sockets> gcc -o TCPEchoServer TCPEchoServer.c DieWithError.c HandleTCPClient.c
milo:~/CS556/sockets> gcc -o TCPEchoClient TCPEchoClient.c DieWithError.c
milo:~/CS556/sockets> TCPEchoServer 3451 &
[1] 6273
milo:~/CS556/sockets> TCPEchoClient 0.0.0.0 olá! 3451
Manipulando cliente 127.0.0.1
Recebido: olá!
milo:~/CS556/sockets> ps
  PID TTY          TEMPO CMD
 5128 pontos/    00:00:00 tchau
 9 6273 pontos/  00:00:00 TCPEchoServer
 9 6279 pontos/9 00:00:00 ps
milo:~/CS556/sockets> matar 6273
milo:~/CS556/sockets>
[1]      Terminado                      Servidor TCPEcho 3451
milo:~/CS556/sockets>
```

O Fim - Perguntas

