

# Programação de Aplicações em Rede usando *Sockets*

## Objetivos:

- Conhecer a API *Sockets*, que permite a programas de aplicação comunicar-se através da Internet

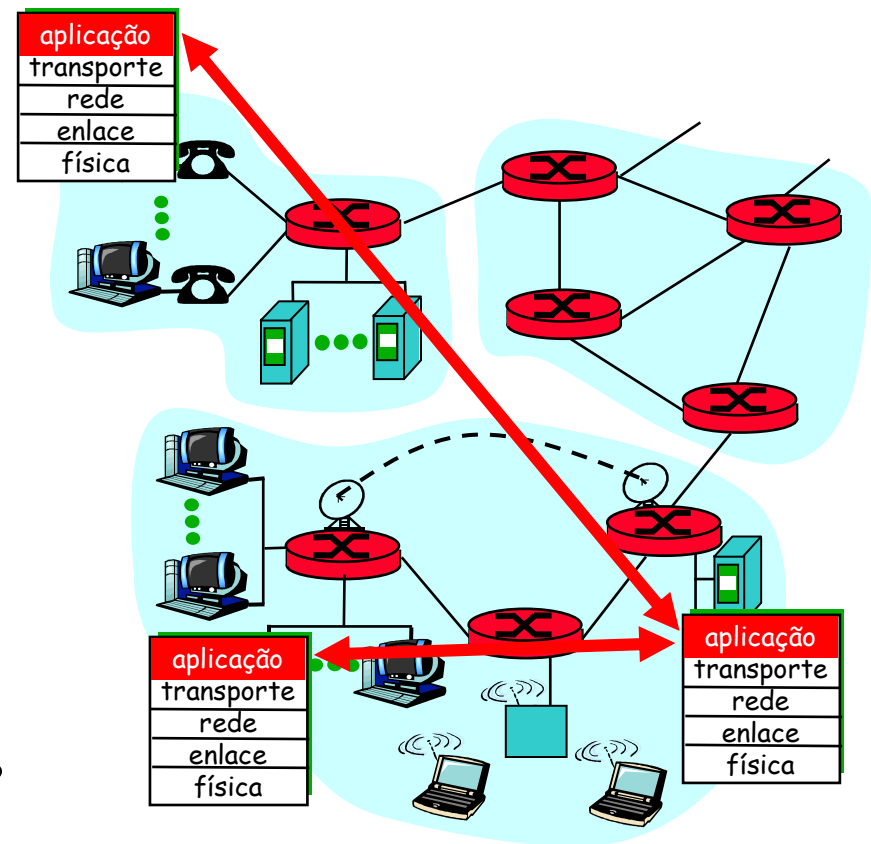
# Aplicações e Protocolo de Aplicação

## Aplicação: processos distribuídos comunicantes

- rodam nos hospedeiros da rede no “espaço do usuário”
- trocam mensagens para realização da aplicação
- e.x., email, ftp, Web

## Protocolos de aplicação

- fazem parte das aplicações
- definem as mensagens trocadas e as ações tomadas
- usam serviços de comunicação das camadas inferiores (TCP, UDP)



# Aplicações de Rede: jargão

**Processo:** programa executando num hospedeiro.

- dentro do mesmo hospedeiro: **comunicação interprocesso** (definida pelo OS).
- processos executando em diferentes hospedeiros se comunicam através de um **protocolo da camada de aplicação**

- **agente do usuário:** processo que interfaceia com o usuário ‘em cima’ e com a rede ‘em baixo’.
  - implementa um protocolo da camada de aplicação
  - Web: *browser*
  - correio-e: leitor de correio
  - *streaming de áudio/vídeo: media player*

# Paradigma Cliente-Servidor

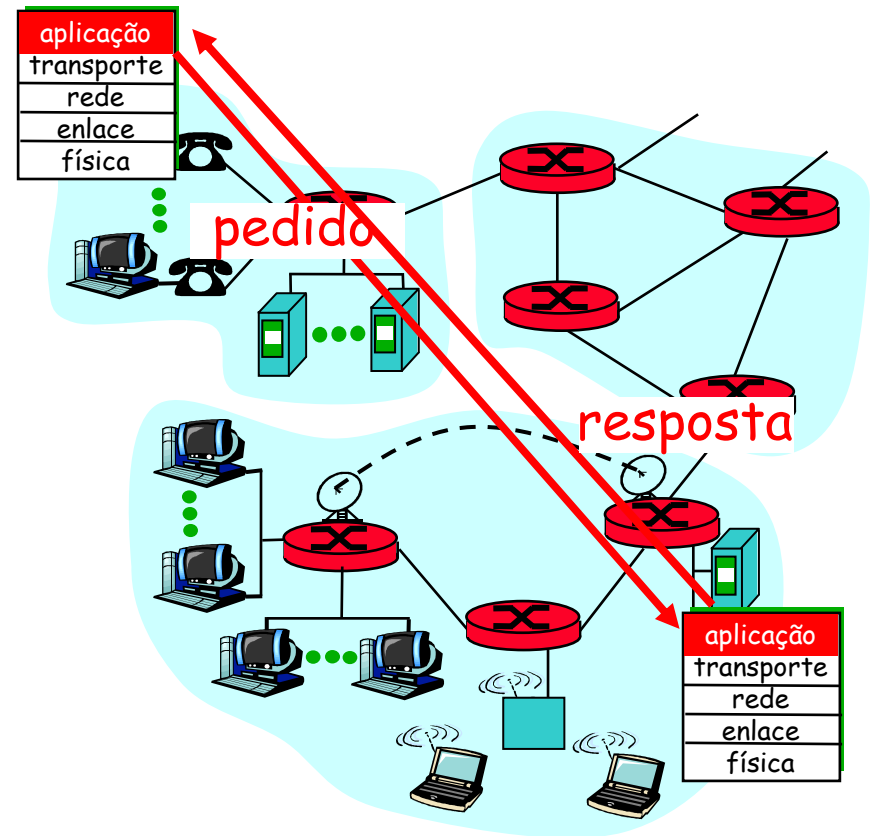
Aplicações de rede típicas têm duas partes: *cliente* e *servidor*

## Cliente:

- inicia a comunicação com o servidor (“fala primeiro”)
- tipicamente solicita serviços do servidor,
- Web: cliente implementado no navegador; correio-e: leitor de correio

## Servidor:

- fornece os serviços solicitados ao cliente
- e.x., servidor Web envia a página Web solicitada, servidor de correio-e envia as mensagens, etc.



# Interfaces de Programação

## *API: application programming interface*

- define a interface entre a camada de aplicação e de transporte
- soquete (*socket*):  
API da Internet
  - dois processos se comunicam enviando dados para o soquete e lendo dados do soquete

Q: Como um processo “identifica” o outro processo com o qual ele quer se comunicar?

- **Endereço IP** do computador no qual roda o processo remoto
- “**número de porta**” - permite ao computador receptor determinar o processo local para o qual a mensagem deve ser entregue.

# Serviços de Transporte da Internet

## serviço TCP:

- *orientado a conexão:* requerido o estabelecimento de conexão entre cliente e servidor
- *transporte confiável* de dados entre os processos emissor e receptor
- *controle de fluxo:* compatibilização de velocidade entre o transmissor e o receptor
- *controle de congestionamento :* protege a rede do excesso de tráfego
- *não oferece:* garantias de temporização e de banda mínima

## serviço UDP:

- transferência de dados não confiável entre os processos transmissor e receptor
- não oferece: estabelecimento de conexão, confiabilidade, controle de fluxo e de congestionamento, garantia de temporização e de banda mínima.

# Aplicações e Protocolos de Transporte da Internet

<b>Aplicação</b>	<b>Protocolo de Aplicação</b>	<b>Protocolo de Transporte</b>
e-mail	smtp [RFC 821]	TCP
acesso a terminal remotos	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
transferência de arquivos	ftp [RFC 959]	TCP
<i>streaming</i> multimídia	RTP ou proprietário (e.g. RealNetworks)	TCP ou UDP
servidor de arquivos remoto	NFS	TCP ou UDP
telefonia Internet	RTP ou proprietário (e.g., Vocaltec)	tipicamente UDP

# A Interface *Socket*

- Acesso à rede através de uma interface semelhante à do sistema de arquivos
- Cada protocolo de transporte oferece um conjunto de serviços. A API Sockets fornece a abstração para acessar esses serviços
- A API define chamadas de funções para criar, fechar, ler e escrever em um soquete.
- O soquete (*socket*) é a abstração usada para comunicação na API *Socket*
  - Define um ponto terminal de comunicação para um processo
  - O SO mantém informação sobre o soquete e a sua conexão
  - A aplicação referencia o soquete para enviar e receber dados, etc



# O que é Necessário para a Comunicação Usando Soquetes?

Quatro parâmetros

- Endereço IP fonte
- Porta fonte
- Endereço IP destino
- Porta destino

Fornecidos no momento da amarração (*binding*) do soquete

# Criação de um Soquete

**int socket(int familia, int tipo, int protocolo)**

- familia:  
PF\_INET ou PF\_UNIX
- tipo:  
SOCK\_STREAM ou  
SOCK\_DGRAM
- protocolo:  
geralmente 0

A chamada retorna um descritor para o soquete (*handle*)

# Amarração de um Soquete

```
int bind ( int socket,  
           struct sockaddr *address,  
           int addr_len)
```

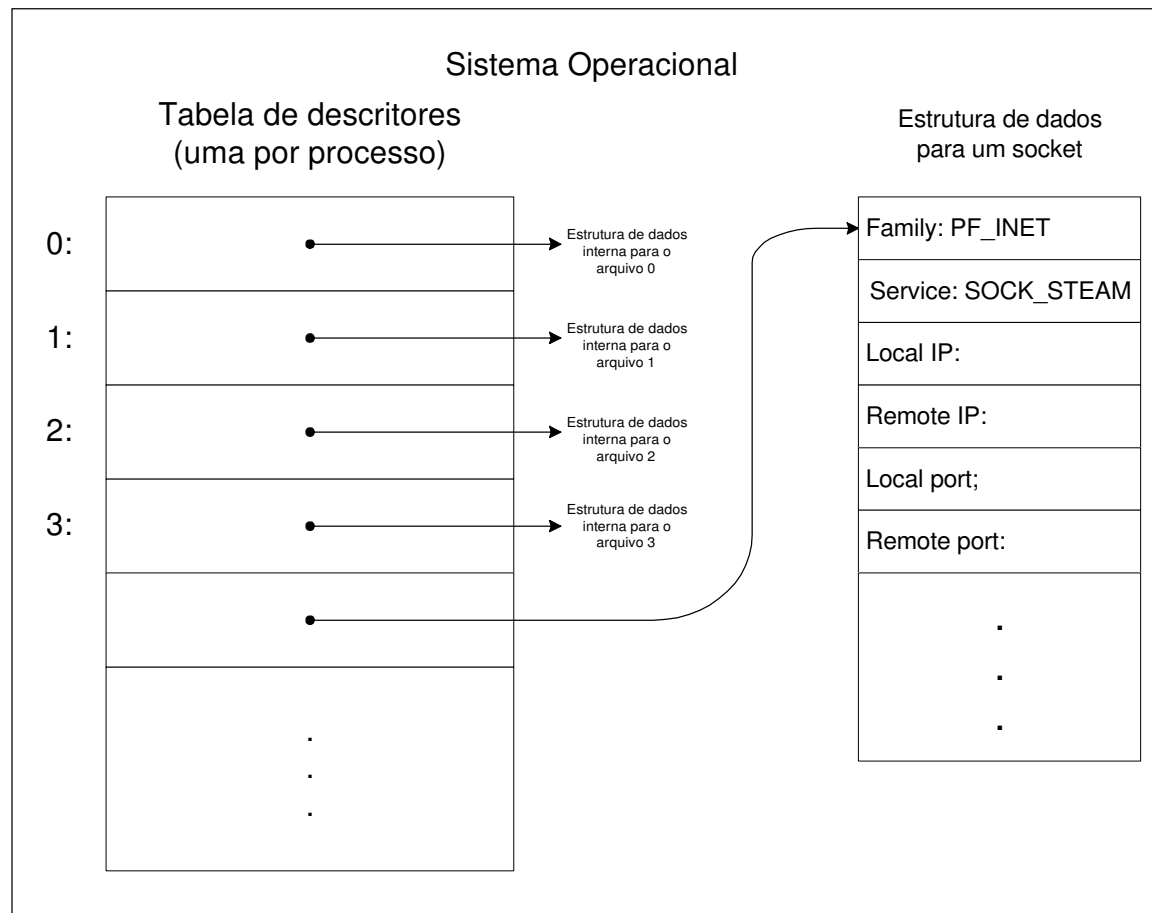
- Executada por um servidor TCP ou UDP
- Amarra um soquete a um endereço local
  - Endereço IP
  - porta

# Estrutura de um Endereço

```
struct sockaddr {          /* struct to hold an address*/
    u_char  sa_len;        /* total length */
    u_short sa_family;     /* type of address */
    char sa_data[14]      /* value of address */
};
```

```
struct sockaddr_in {      /* struct to hold an address */
    u_char  sin_len        /* total length */
    u_short sin_family;    /* type of address */
    u_short sin_port;      /* port protocol number */
    struct  in_addr in_addr sin_addr;
    /* IP address (declared to be u_long type on some systems */
    char sin_zero[8]      /* unused (set to zero) */
};
```

# Estrutura de Dados do Sistema para *sockets*



# Servidor TCP: Listen

**int listen (int socket, int queuelen)**

- Especifica o número máximo de pedidos pendentes de conexão para um soquete
- Quando o servidor estiver processando uma conexão, *queulen* conexões podem estar pendentes numa fila

# Servidor TCP: Conexão Passiva (accept)

```
int accept (int socket,  
            struct sockaddr *address,  
            int *addr_len)
```

- Retorna apenas quando um cliente remoto requisitar uma conexão
- *\*address*: endereço do cliente
- Retorna um descritor para a conexão estabelecida

# Cliente TCP: Conexão Ativa (connect)

```
int connect ( int socket,  
             struct sockaddr *address,  
             int *addr_len)
```

- **\*address: endereço remoto (do servidor)**
- **Retorna apenas quando a conexão for estabelecida**
- **Ao retorno, o soquete *socket* estará conectado e pronto para a comunicação**



# Soquetes: resumo

## **Cliente:**

- int socket(int domain, int type, int protocol)
- int connect ( int socket, struct sockaddr \*address, int addr\_len)

## **Servidor:**

- int socket(int domain, int type, int protocol)
- int bind (int socket, struct sockaddr \*address, int addr\_len)
- int listen (int socket, int backlog)
- int accept (int socket, struct sockaddr \*address, int \*addr\_len)

# Comunicação

```
int send (int socket, char *message, int msg_len, int flags)  
    // TCP
```

```
int sendto (int socket, void *msg, int len, int flags,  
    struct sockaddr * to, int tolen ); //UDP)
```

```
int write(int socket, void *msg, int len); // TCP
```

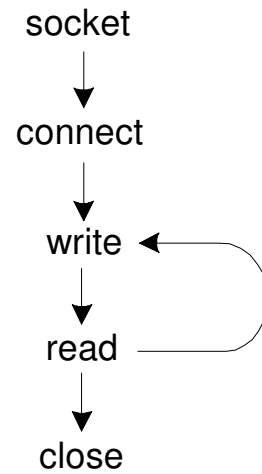
```
int recv (int socket, char *buffer, int buf_len, int flags) //TCP
```

```
int recvfrom(int socket, void *msg, int len, int flags,  
    struct sockaddr *from, int *fromlen); // UDP
```

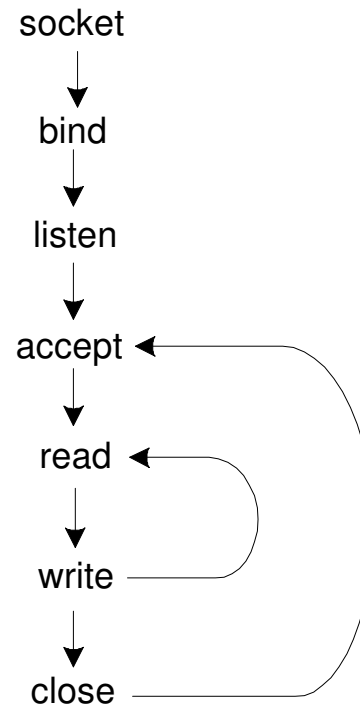
```
int read(int socket, void *msg, int len); //TCP
```

# Uso das Primitivas

LADO CLIENTE



LADO SERVIDOR



# Conversão de Inteiros

- Ordem de bytes da rede: byte mais significativo primeiro
  - padrão TCP/IP para inteiros usados nos cabeçalhos dos protocolos

htons(): Host-to-network byte order for a short word (2 bytes)

htonl(): Host-to-network byte order for a long word (4 bytes)

ntohs(): Network-to-host byte order for a short word

ntohl(): Network-to-host byte order for a long word

# Outras Funções Utilitárias

gethostname() -- get name of local host

getpeername() -- get address of remote host

getsockname() -- get local address of socket

getXbyY() -- get protocol, host, or service number using known number, address, or port, respectively

getsockopt() -- get current socket options

setsockopt() -- set socket options

ioctl() -- retrieve or set socket information

inet\_addr() -- convert “dotted” character string form of IP address to internal binary form

inet\_ntoa() -- convert internal binary form of IP address to “dotted” character string form

# Bibliografia

- COMER, D. E., STEVENS, D. L.,  
Internetworking With TCP/IP Volume III: Client-Server Programming and Applications, Linux/POSIX Socket Version, Prentice-Hall International 2001.
- STEVENS, W.R.; "UNIX NETWORKING PROGRAMMING – Networking APIs: Sockets and XTI – Volume 1 – Second Edition . Prentice Hall PTR – 1998.