

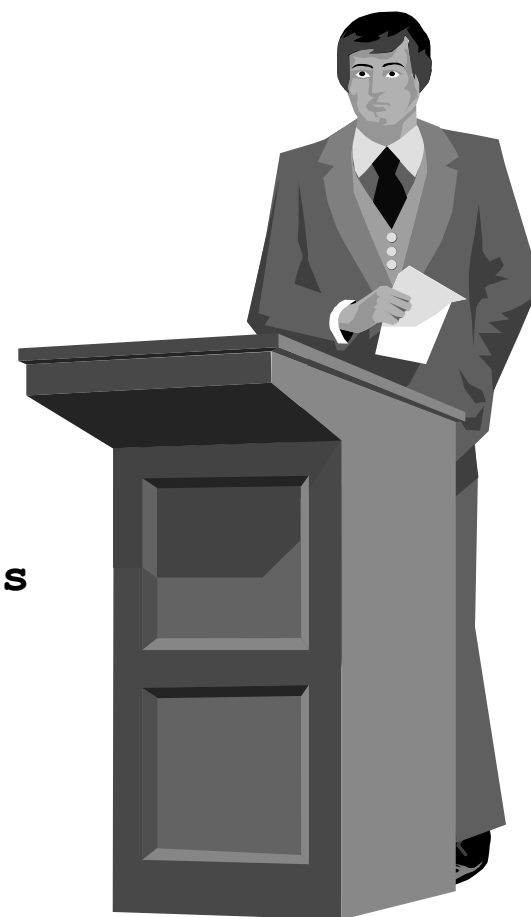
Cliente UDP

Volnys Borges Bernal

`volnys@lsi.usp.br`

Departamento de Sistemas Eletrônicos

Escola Politécnica da USP



Agenda

- **Resumo das chamadas sockets para UDP**

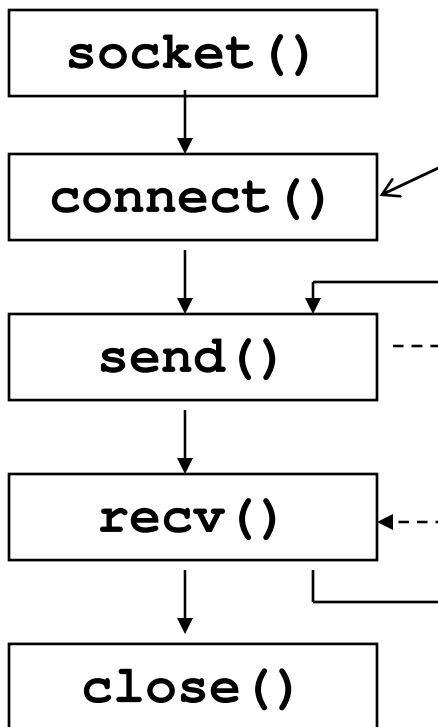
- **Uso das chamadas sockets para UDP**
 - ❖ Chamada socket()
 - ❖ Chamada connect()
 - ❖ Chamada send()
 - ❖ Chamada recv()
 - ❖ Chamada close()

Resumo das chamadas UDP



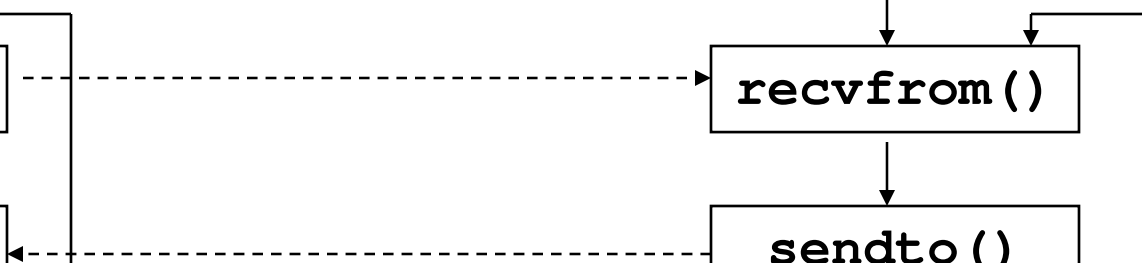
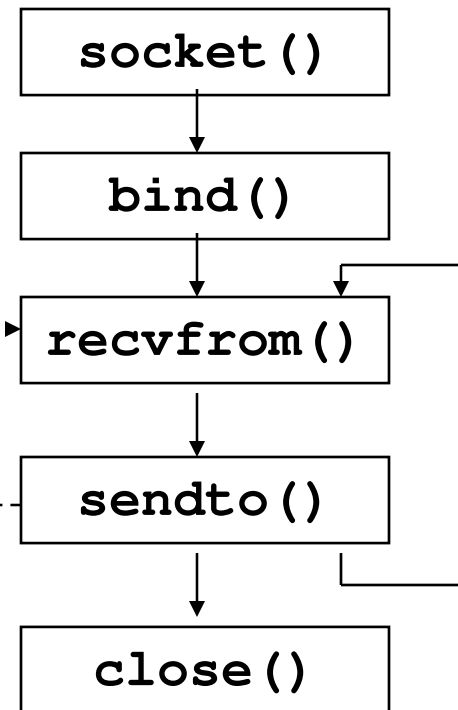
Resumo de Chamadas UDP

Lado Cliente



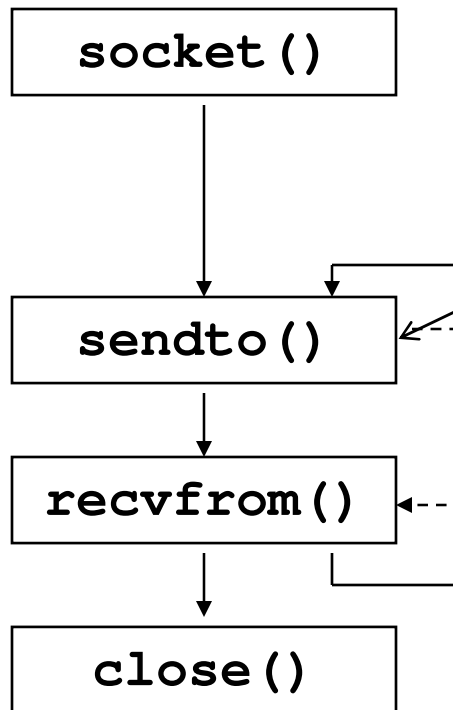
*Pré define o
parceiro de
comunicação para
todo send()*

Lado Servidor



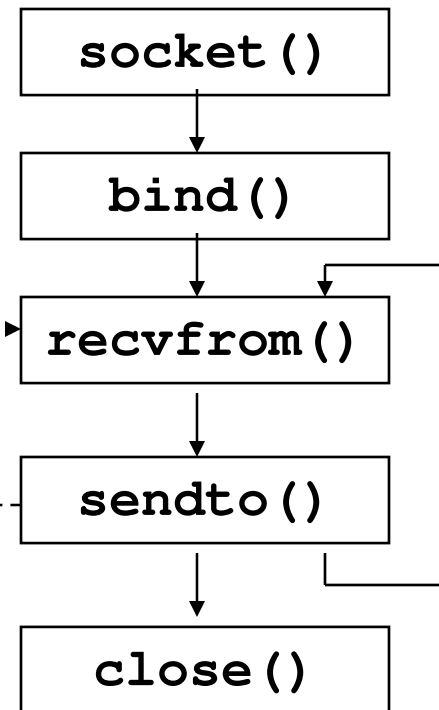
Resumo de Chamadas UDP

Lado Cliente

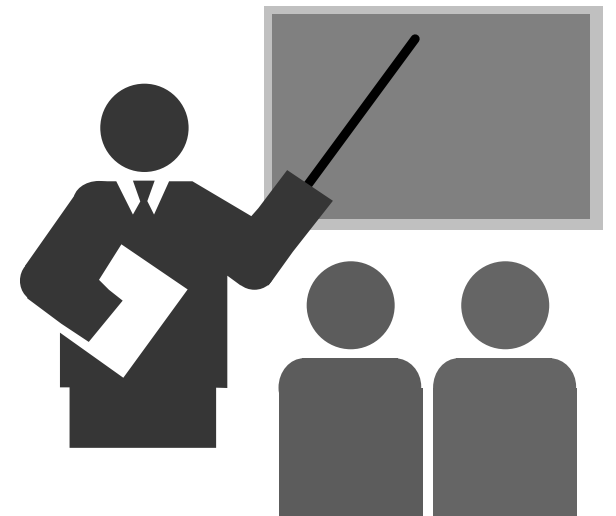


*Informa o
parceiro de
comunicação a
cada chamada*

Lado Servidor



Chamada socket()



Chamada socket()

❑ Objetivo

- ❖ Criar um novo socket (plug de comunicação). Aloca estruturas de dados no sistema operacional para suportar a comunicação.

❑ Resultado

- ❖ Retorna o descritor de arquivo (número inteiro).

❑ Sintaxe

```
sd = socket (int domain, int type, int protocol)
```

❑ Observação:

- ❖ Quando um socket é criado, não possui nenhuma informação sobre o parsocket (endereços IPs e portas dos parceiros).

Chamada socket()

□ Sintaxe geral

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

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

*Socket
descriptor*

Para PF_INET usar 0

Pilha de protocolos:

- *PF_LOCAL* (file)
- *PF_INET* (IPv4)
- *PF_INET6* (IPv6)
- *PF_X25* (X25)

Tipo da comunicação:

- *SOCK_STREAM* (TCP)
- *SOCK_DGRAM* (UDP)
- *SOCK_RAW* (IP)

Chamada socket()

□ Tipo de serviço

❖ SOCK_STREAM

- Para ser utilizado com o protocolo TCP
- Canal de comunicação full duplex
- Fluxo de bytes sem delimitação
- Chamadas para transmissão e recepção de dados:
 - read(), write() ou send(), recv()

❖ SOCK_DGRAM

- Para ser utilizado com o protocolo UDP
- Datagrama (mensagens)
- Chamadas para transmissão e recepção de dados:
 - send(), sendto(), recv() ou recvfrom()

❖ SOCK_RAW

- Permite acesso a protocolos de mais baixo nível
- Datagrama (mensagens)
- Chamadas para transmissão e recepção de dados:
 - send(), recv()

Chamada socket()

❑ *Para criar um socket TCP*

```
#include <sys/socket.h>  
sd = socket(AF_INET, SOCK_STREAM, 0);
```

❑ *Para criar um socket UDP*

```
#include <sys/socket.h>  
sd = socket(AF_INET, SOCK_DGRAM, 0);
```

Chamada socket()

❑ Exemplo de criação de socket UDP

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

int sd; // socket descriptor
. . .
sd = socket(PF_INET, SOCK_DGRAM, 0);
if (sd == -1)
    {
    perror("Erro na chamada socket");
    exit(1);
    }
. . .
```

Chamada Connect()



Chamada connect()

□ Objetivo

- ❖ Estabelecer uma sessão de comunicação TCP, UDP ou IP

□ Detalhamento

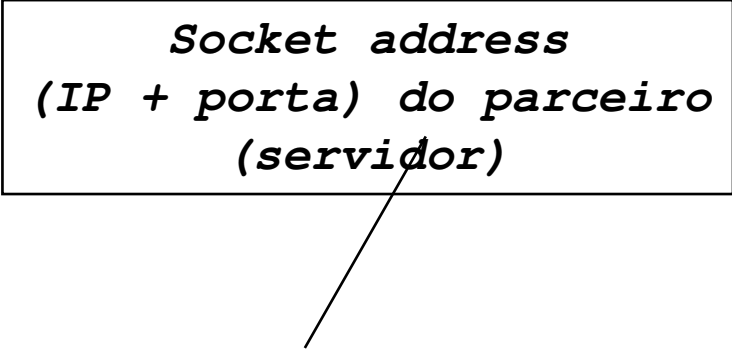
- ❖ Deve ser utilizado somente no lado cliente
- ❖ UDP:
 - Informa ao sistema operacional o socket address (IP+porta) do parceiro de comunicação
 - Não são enviados datagramas
- ❖ TCP:
 - Informa ao sistema operacional o socket address (IP+porta) do parceiro de comunicação
 - Estabelece a conexão TCP (*3 way handshake*)

Chamada connect()

□ Sintaxe

```
#include <netdb.h>
int connect(int sd,
            struct sockaddr *serversockaddr,
            int size)
```

*Socket address
(IP + porta) do parceiro
(servidor)*



*Socket
descriptor*



*Tamanho da estrutura de
endereço (sockaddr_in)*



Chamada connect()

```
#include <netdb.h>

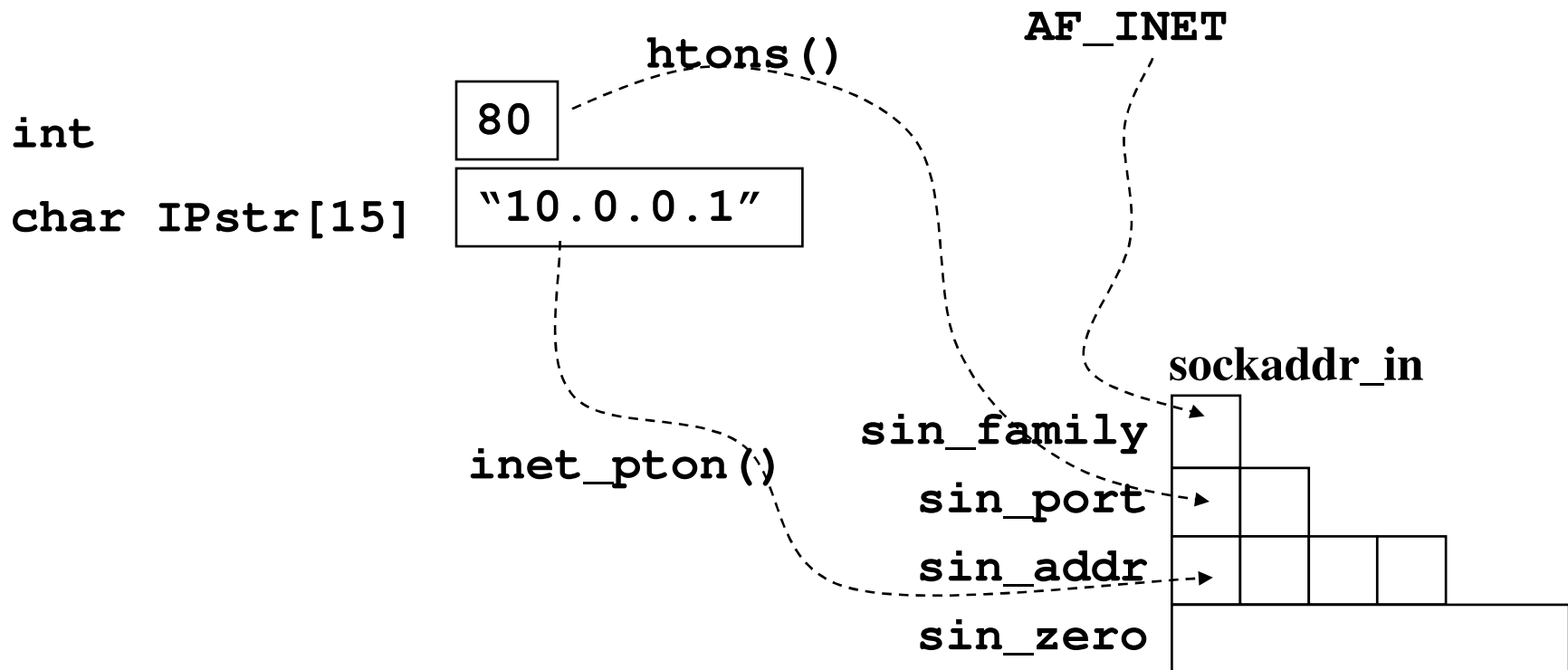
int          status;          //estado da chamada
struct sockaddr_in  serveraddr; //endereço do servidor
...

// define endereço destino
serveraddr.sin_family = AF_INET;
serveraddr.sin_port   = htons(serverport);
status = inet_pton(AF_INET, stringIP, &serveraddr.sin_addr);
if (status <= 0)
    perror("Erro na conversão do endereço IP");

// ativa connect
status = connect( sd,
                 (struct sockaddr *)&serveraddr,
                 sizeof(serveraddr) );
if (status != 0)
    perror("Erro na chamada connect");
```

Chamada connect

- Exemplo de iniciação da estrutura `sockaddr_in`

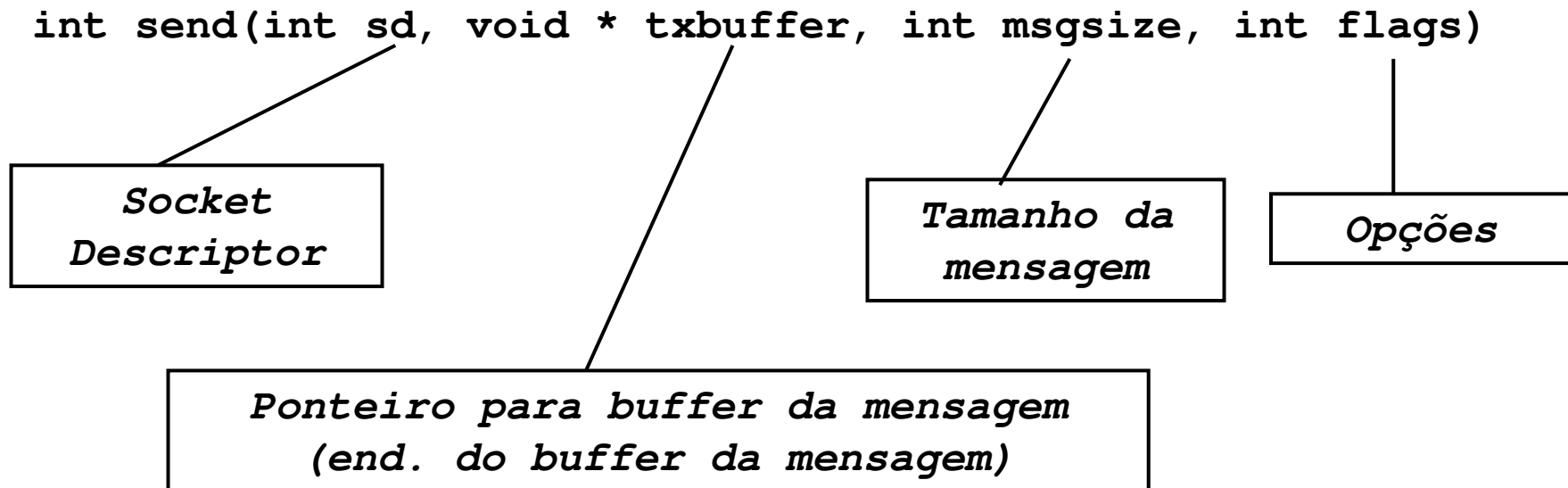


Chamada send()



Chamada send()

- ❑ Função para transmissão de dados
- ❑ Pode ser utilizada por clientes e servidores



Chamada send()

□ Exemplo:

```
char txbuffer[80];  
  
. . .  
status = send (sd, txbuffer, strlen(txbuffer)+1, 0)  
if (status < 0)  
    perror("Erro na chamada send");  
. . .
```

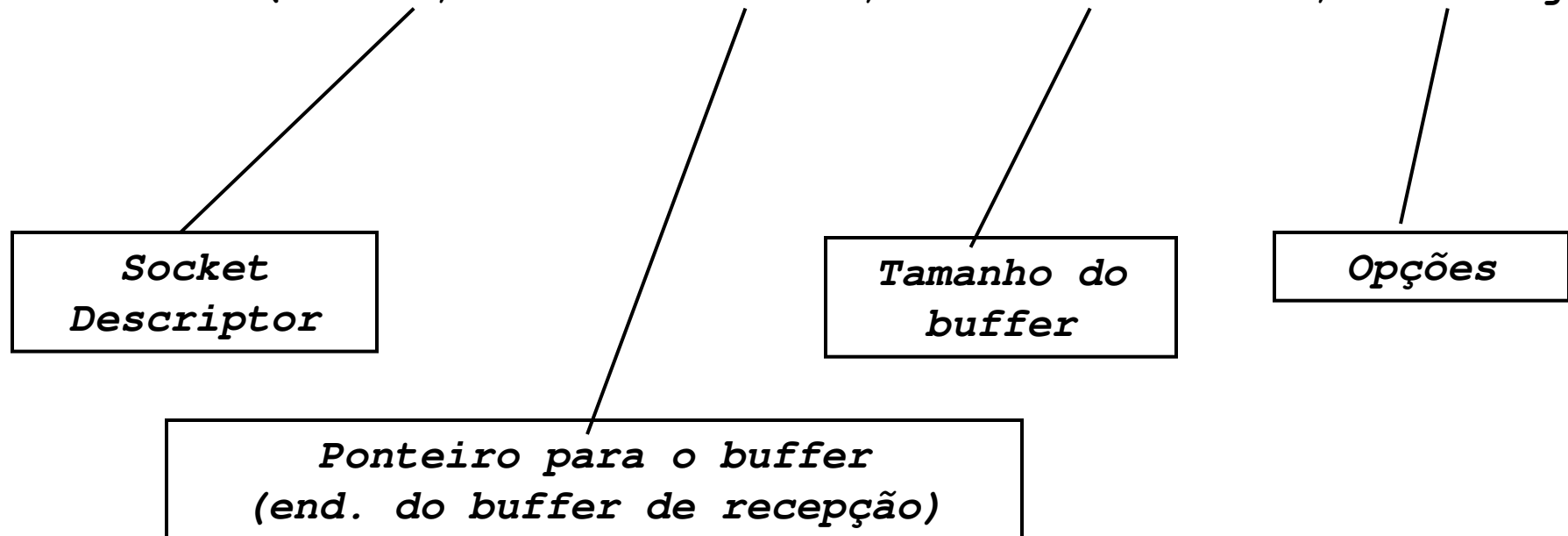
Chamada recv()



Chamada recv()

- ❑ Recebimento de datagramas
- ❑ Pode ser utilizada por clientes e servidores

```
int recv(int sd, void * rxbuffer, int rxbuffersize, int flags)
```



Chamada recv()

□ Exemplo:

```
char rxbuffer[80];  
  
. . .  
status = recv(sd, rxbuffer, sizeof(rxbuffer), 0)  
if (status < 0)  
    perror("Erro na chamada recv");  
printf("MSG recebida: %s\n", rxbuffer);  
. . .
```

Chamada recv()

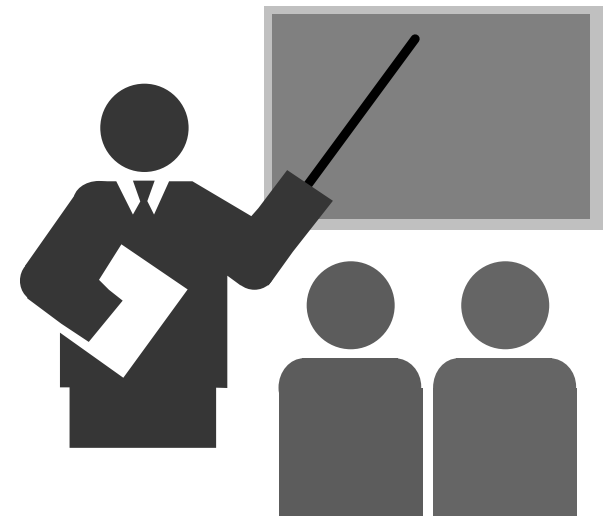
❑ Bloqueante

- ❖ Se não existirem mensagens na fila de recepção o processo fica aguardando sua chegada
- ❖ Exceção: quando o socket for criado como não bloqueante (ver `fcntl(2)`).

❑ Retorno

- ❖ Se a chamada tiver sucesso, o valor retornado é o tamanho do datagrama

Chamada close()



Chamada close()

❑ Objetivo

- ❖ Fechar o descritor de arquivos (neste caso, fecha o socket).
- ❖ Se ainda existirem dados para serem transmitidos pelo socket, aguarda por alguns segundos a finalização desta transmissão.

❑ Resultado

- ❖ Fecha o descritor do arquivo.

❑ Sintaxe

```
int close (int sd)
```

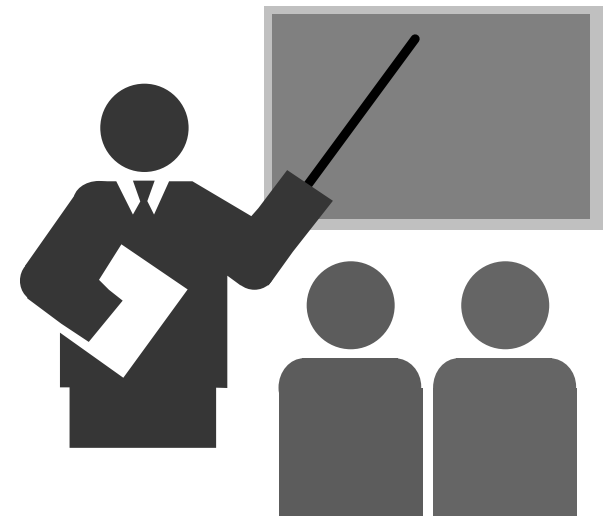
Chamada close()

□ Exemplo:

```
int sd; // socket descriptor
. . .

status = close(sd);
if (status == -1)
    perror("Erro na chamada close");
. . .
```

Exercício



Exercício

- (1) Identifique a porta utilizada no serviço “echo”.**

- (2) Implemente um cliente para o serviço “echo” utilizando o protocolo UDP.**
 - ❖ O serviço echo responde exatamente com a seqüência ASCII recebida.

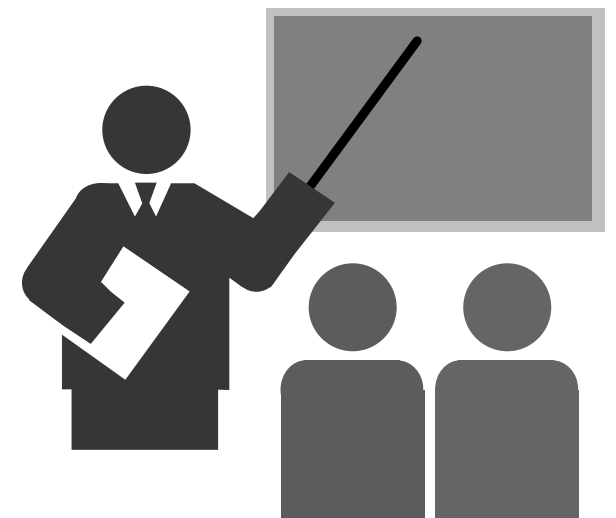
Exercício

(3) Identifique a porta utilizada no serviço “daytime”.

(4) Implemente um cliente para o serviço daytime utilizando o protocolo UDP.

- ❖ O serviço daytime UDP responde com a data e hora do servidor no instante de recebimento do datagrama UDP.

Referências Bibliográficas



Referências Bibliográficas

- ❑ **COMMER, DOUGLAS; STEVENS, DAVID**
 - ❖ Internetworking with TCP/IP: volume 3: client-server programming and applications
 - ❖ Prentice Hall
 - ❖ 1993