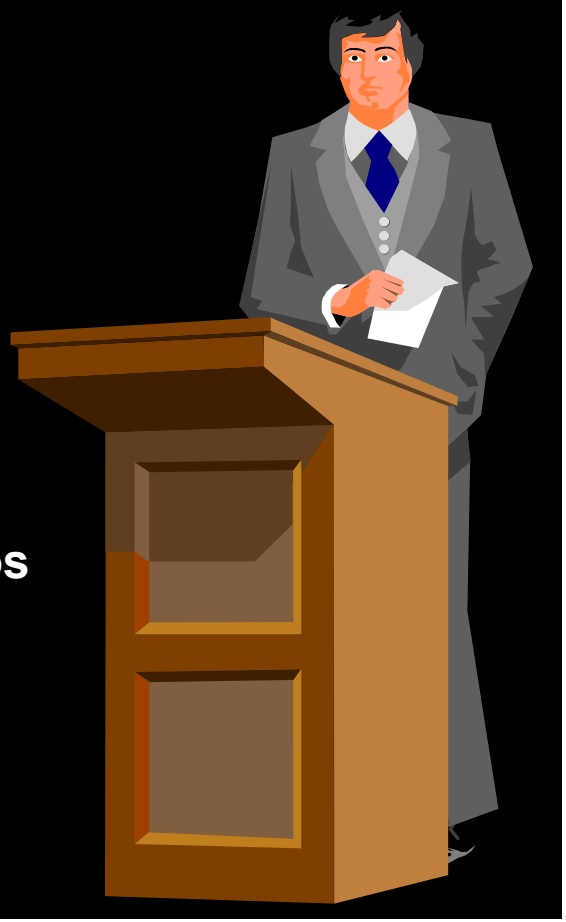
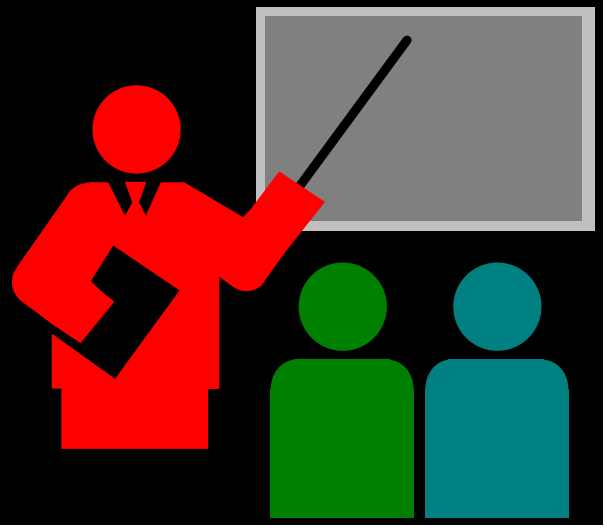
Servidor TCP

Volnys Borges Bernal

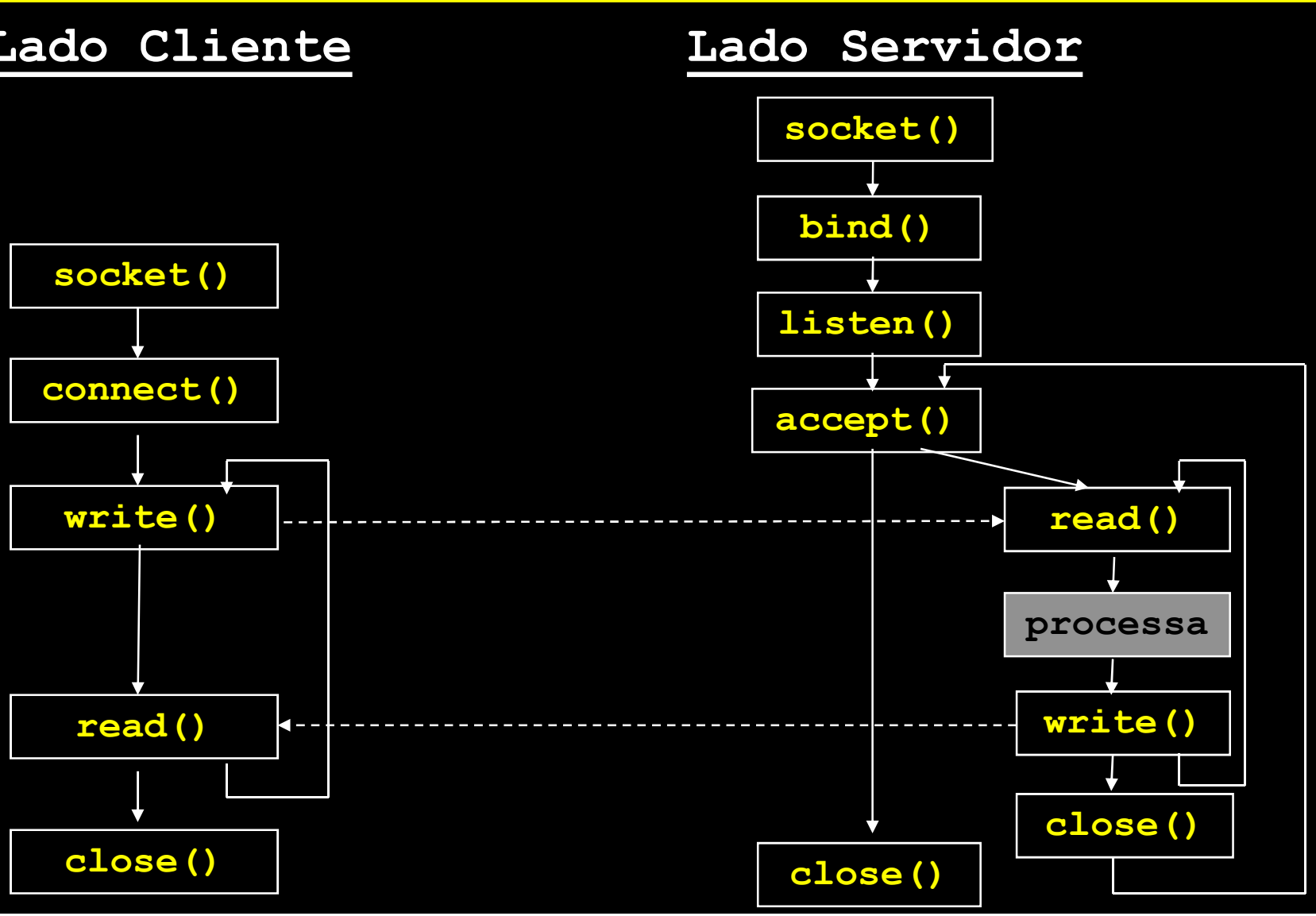
Depto de Engenharia de Sistemas Eletrônicos
Escola Politécnica da USP



Resumo das Chamadas TCP



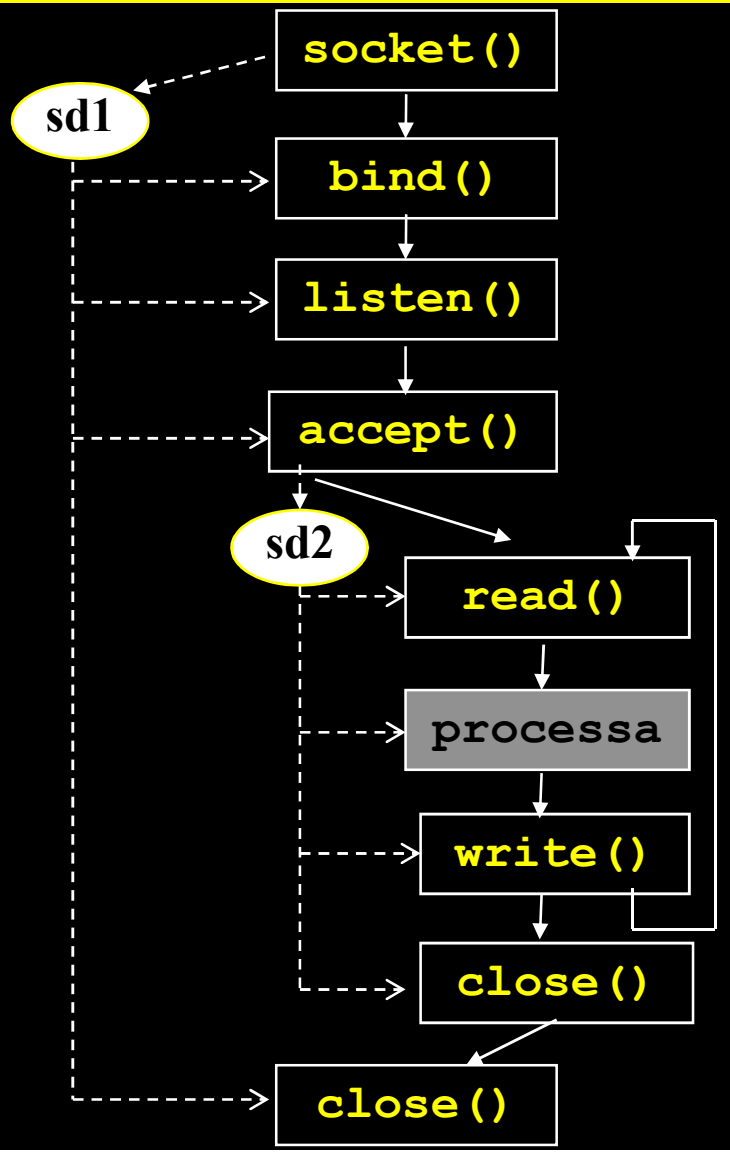
Resumo das Chamadas TCP



Resumo das Chamadas TCP

Detalhamento
Lado Servidor

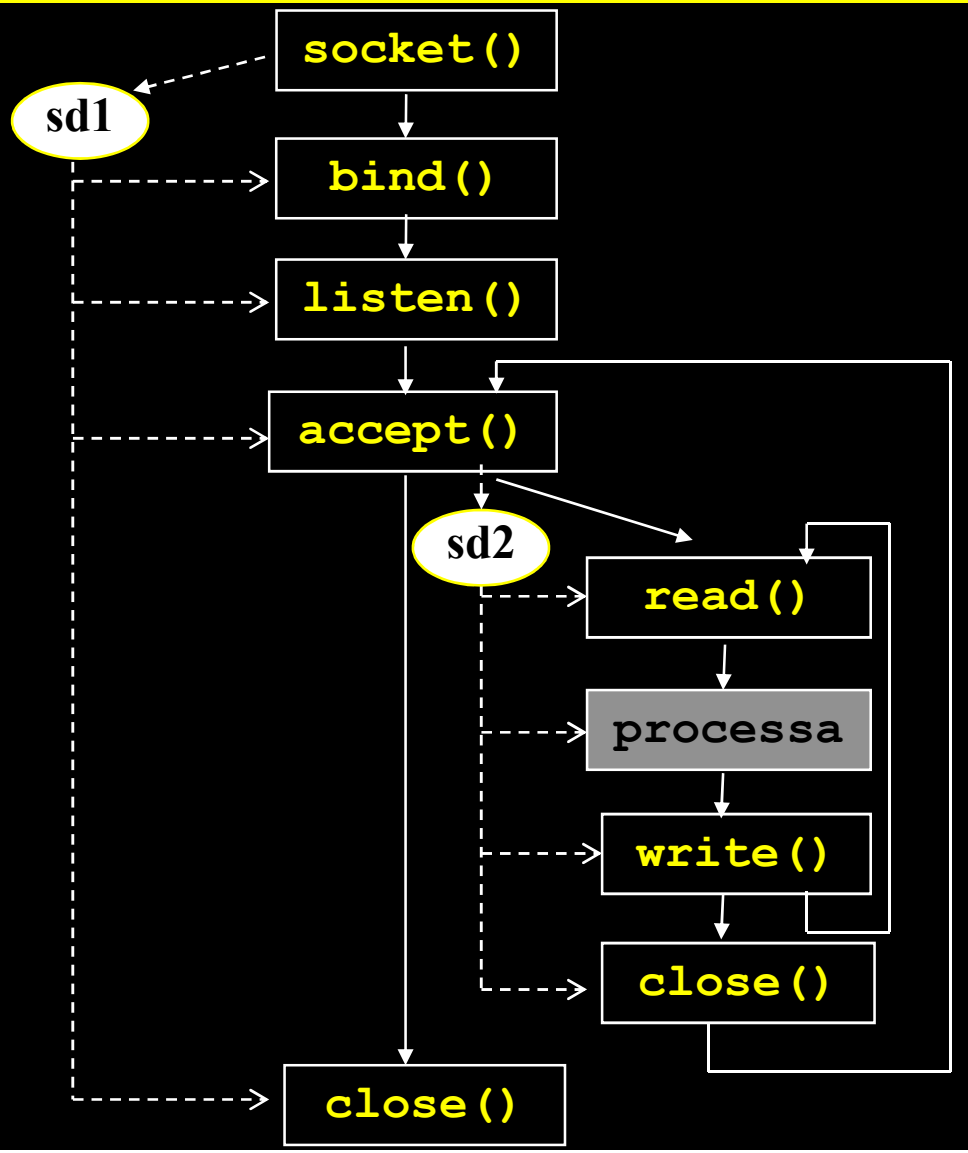
Atende somente
uma conexão.



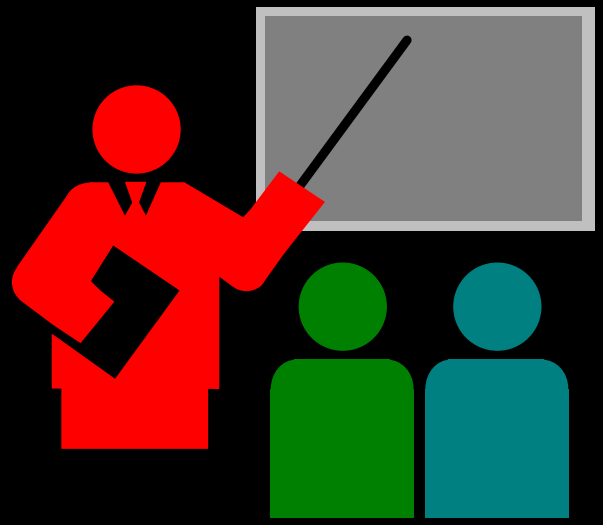
Resumo das Chamadas TCP

Detalhamento
Lado Servidor

Atende várias conexões, uma de cada vez.



Chamada socket()



Chamada socket()

- **Objetivo**
  - Criar um novo socket (plug de comunicação).
  - Aloca estruturas de dados no sistema operacional para suportar a sessão de comunicação.
- **Resultado**
  - Retorna o descritor de arquivo (número inteiro).
- **Sintaxe**

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

- **Observação:**
  - Após a criação do socket, não existe nenhuma informação sobre o parsocket (endereços IPs e portas dos participantes). Outras chamadas são responsáveis por associar estas informações.

Chamada socket()

- **Sintaxe geral**

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol)
```

Socket descriptor

Para PF_INET use 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 trasnmissão e recepção de dados:
      - send(), sendfrom(), recv() ou recvfrom()
  - SOCK_RAW
    - Permite acesso a protocolos de mais baixo nível
    - Datagrama (mensagens)
    - Chamadas para trasnmissão 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);
```

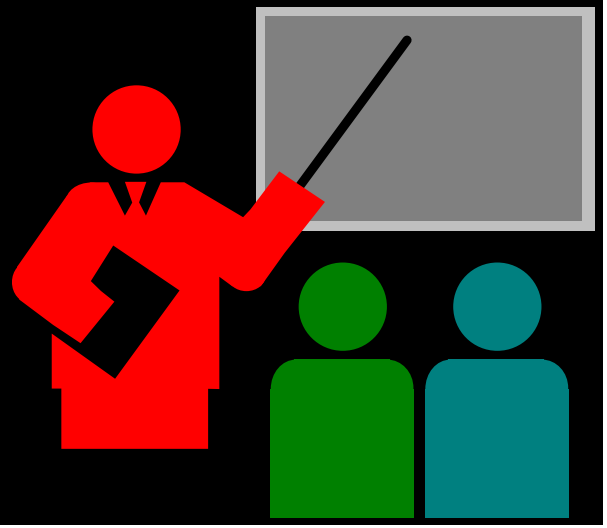
Chamada socket()

- Exemplo de criação de socket TCP

```
#include <sys/socket.h>
int sd;

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

Chamada bind()



Chamada bind()

- **Objetivo**
  - Definir o *socket address* local (endereço IP e porta TCP).
  - Deve ser utilizado somente no lado servidor.

- **Sintaxe**

```
int bind(int sd, struct sockaddr *myaddr, socklen_t addrlen)
```

- **Valor retornado pela função:**
  - –1: erro
  - 0: sucesso

Chamada bind()

- Sintaxe

```
#include <netdb.h>

int     bind(
             int sd,
             struct sockaddr *myaddr,
             socklen_t addrlen)
```

Socket addres local: Endereço IP da interface local e e porta TCP pela qual o servidor TCP irá aguardar conexões.

Resultado da chamada: sucesso ou erro

Descritor do socket

Tamanho da estrutura de endereço (sockaddr)

Chamada bind()

- **O endereço IP do socket address do servidor pode ser:**
  - Aquele associado a uma das interfaces do servidor:
    - Deve ser informado o endereço IP da interface
  - Todas interfaces locais
    - Deve ser utilizada a macro INADDR_ANY

Chamada bind()

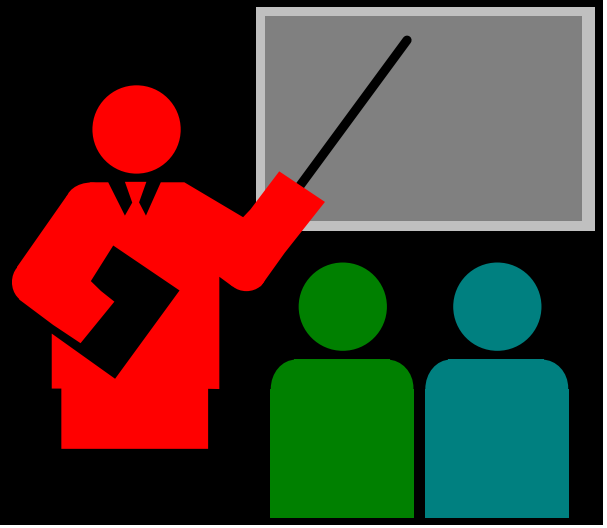
```
#include <netdb.h>

struct sockaddr_in mylocal_addr

. . .
mylocal_addr.sin_family      = AF_INET;
mylocal_addr.sin_addr.s_addr = INADDR_ANY;
mylocal_addr.sin_port        = htons(myport);

status = bind(socketdescriptor,
              (struct sockaddr *) &mylocal_addr,
              sizeof(struct sockaddr_in));
if (status == -1)
    perror("Erro na chamada bind");
```

Chamada listen()



Chamada listen()

- **Objetivo**
  - Abrir o socket address no qual o servidor irá aguardar os pedidos de conexão TCP
  - A partir deste momento os pedidos de conexão serão atendidos.
  - Obs: Uma conexão estabelecida poderá ser conhecida somente após a ativação de accept(). Somente após a chamada accept() conexão pode ser encerrada com close().

- **Sintaxe**

```
int listen(int sd, int queuelenght)
```

- **Valor retornado pela função**
  - 0: sucesso
  - -1: erro

Chamada listen()

- Sintaxe

Resultado da função:
0: sucesso
-1: erro

Descritor do socket

```
int listen(int sd,
           int queuelenght)
```

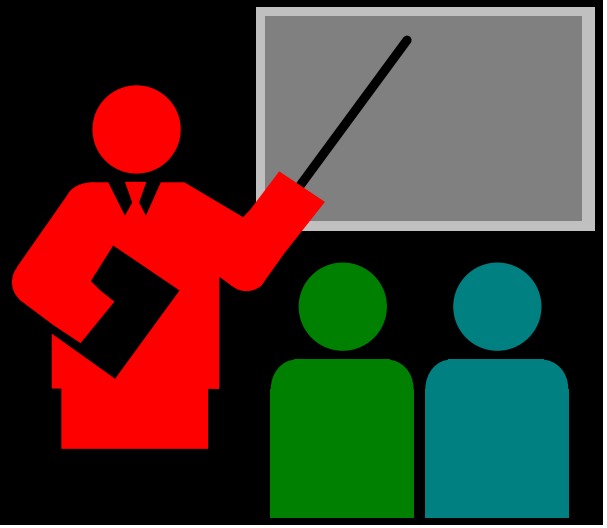
Qde máxima de conexões pendentes sem aceite

Chamada listen()

- Exemplo:

```
#define QLEN 10
...
status = listen(sd,QLEN);
if (status != 0)
        {
        perror("Erro na chamada listen()");
        exit(1);
        }
```

Chamada accept()



Chamada accept()

- **Objetivo**
  - Accept() extrai a primeira conexão da fila e gera um novo *socket descriptor* para esta conexão.
  - O *socket descriptor* original não é afetado por esta chamada.
  - Um dos parâmetros retorna o *socket address* do cliente

- **Sintaxe**

```
int accept (int sd, struct sockaddr *addr,
                                    socklen_t *addrlen)
```

- **Valor retornado pela função**
  - -1: erro
  - Valor não negativo: sucesso, sendo este o valor do novo socket descriptor

Chamada accept()

- Sintaxe

Valor retornado:
Positivo: sucesso, correspondendo ao valor do novo socket
-1: erro

Descritor do socket

```
int accept (int sd,
            struct sockaddr *addr,
            socklen_t *addrlen)
```

Tamanho da estrutura socketaddr (precisa ser uma variável !)

Ponteiro para uma estrutura sockaddr que conterá o endereço (socket address) do parceiro de comunicação

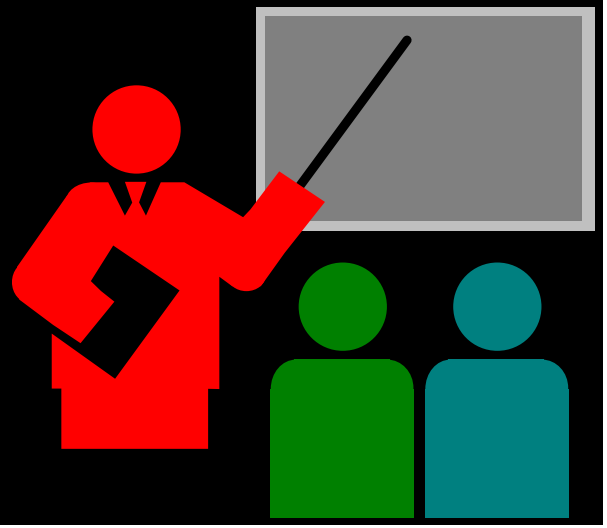
Chamada accept()

- Exemplo

```
int                 newsd;
int                 size;
struct sockaddr_in clientaddr;

....
size = sizeof(clientaddr);
newsd = accept( sd,
                (struct sockaddr *) &clientaddr,
                (socklen_t *)       &size);
if (newsd < 0)
        {
        perror("Erro na chamada accept()");
        exit(1);
        }
....
```

Chamada read()



Chamada read()

- **Objetivo**
  - Recepção/leitura de dados de um descritor
    - Descritor: descritor sockets, descritor de arquivo, ...
  - Pode ser utilizada por cliente ou servidor

- **Sintaxe**
  - `int read(int sd, void *buf, int buffersize)`

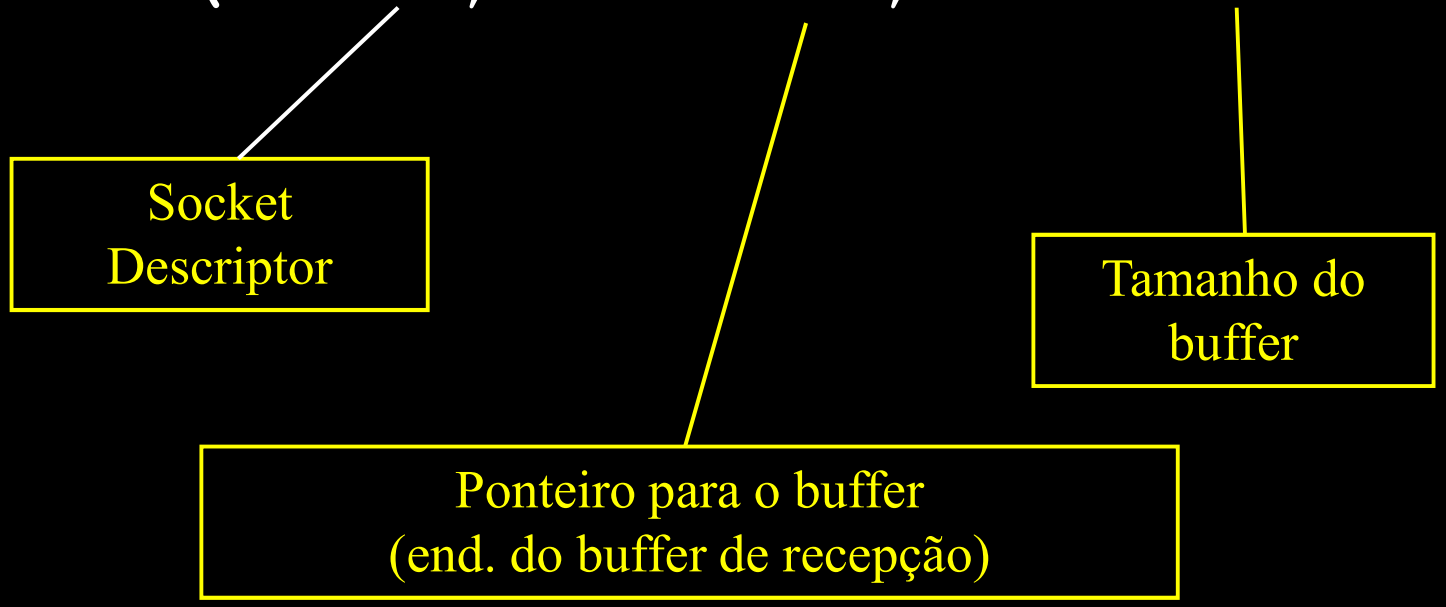
- **Valor retornado pela função**
  - >0: quantidade de bytes lidos
  - 0: end of file
  - -1: erro

Chamada read()

- **Sintaxe:**

```
#include <unistd.h>

int read(int sd, void *buf, int buffersize)
```



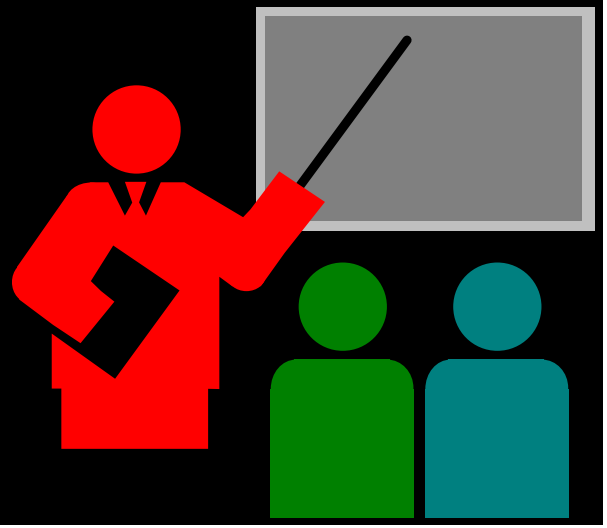
Chamada read()

- Exemplo:

```
char rxbuffer[80];

. . .
status = read(newsd,rxbuffer,sizeof(rxbuffer));
if (status == -1)
    perror("Erro na chamada read");
printf("MSG recebida: %s\n",rxbuffer);
. . .
```

Chamada write()



Chamada write()

- **Objetivo**
  - Transmissão/escrita de dados em um descritor
    - Descritor: descritor sockets, descritor de arquivo, ...
  - Pode ser utilizada por cliente ou servidor

- **Sintaxe**

```
int write(int sd, void *buf, int count)
```

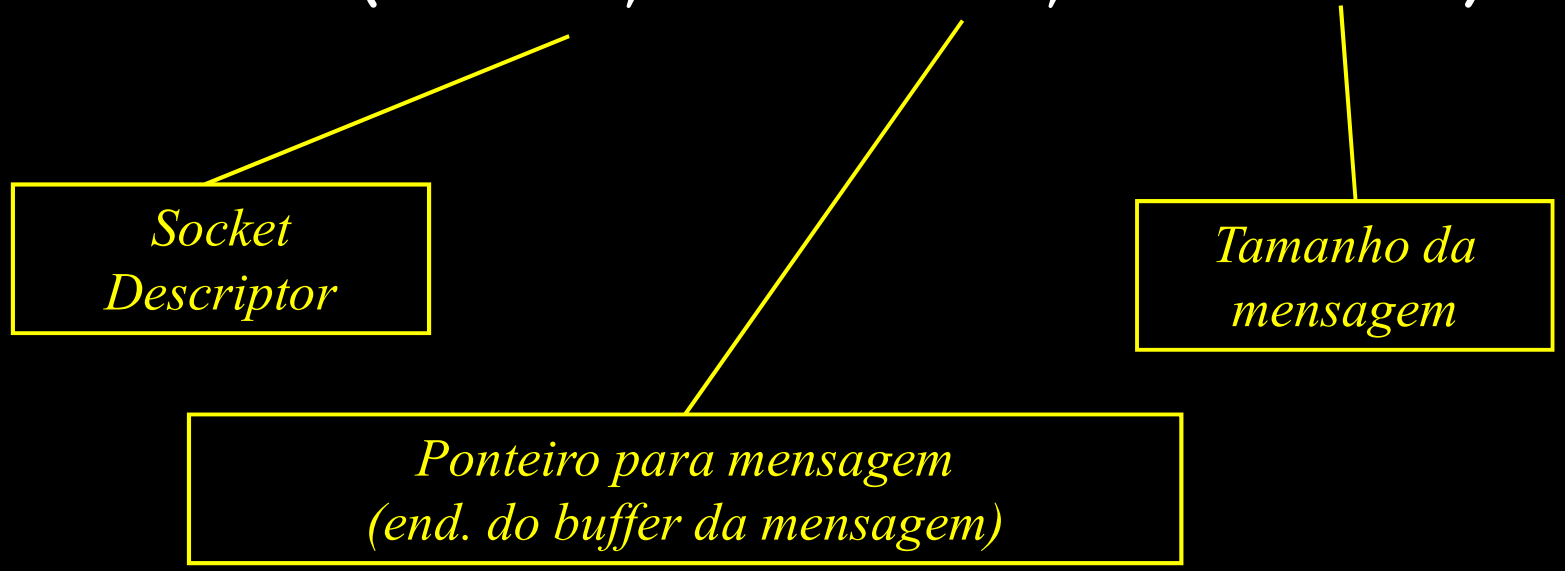
- **Valor retornado pela função**
  - Positivo: quantidade de bytes escritos
  - -1: erro

Chamada write()

- Sintaxe

```
#include <unistd.h>

int write(int sd, void *buf, int count)
```



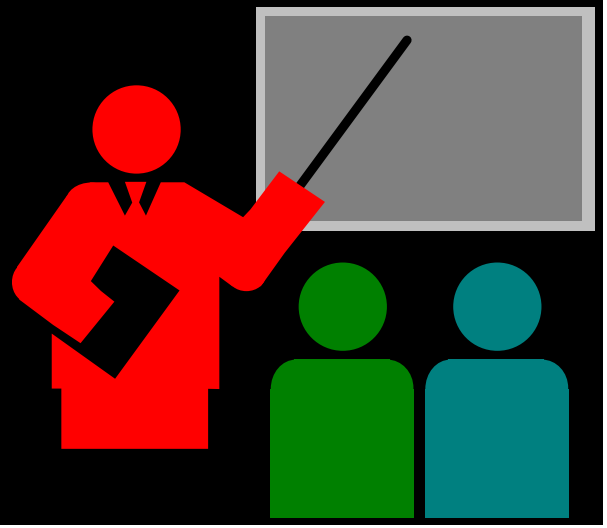
Chamada write()

- Exemplo:

```
#include <unistd.h>
char txbuffer[80];

. . .
status = write(newsd,txbuffer,strlen(txbuffer)+1)
if (status == -1)
  perror("Erro na chamada write");
. . .
```

Chamada close()



Chamada close()

- Objetivo
  - Permite fechar o socket (assim como é feito com arquivo)
  - Lembrar que podem existir vários sockets abertos. É importante fechar todos os sockets.

- Sintaxe

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

- **Valor retornado**

  0 sucesso

  -1 erro

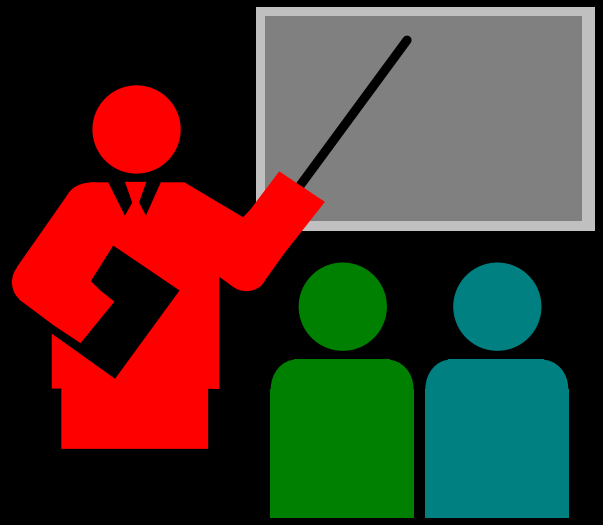
Chamada close()

- Exemplo:

```
int  sd;  // socketdescriptor

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

Exercício

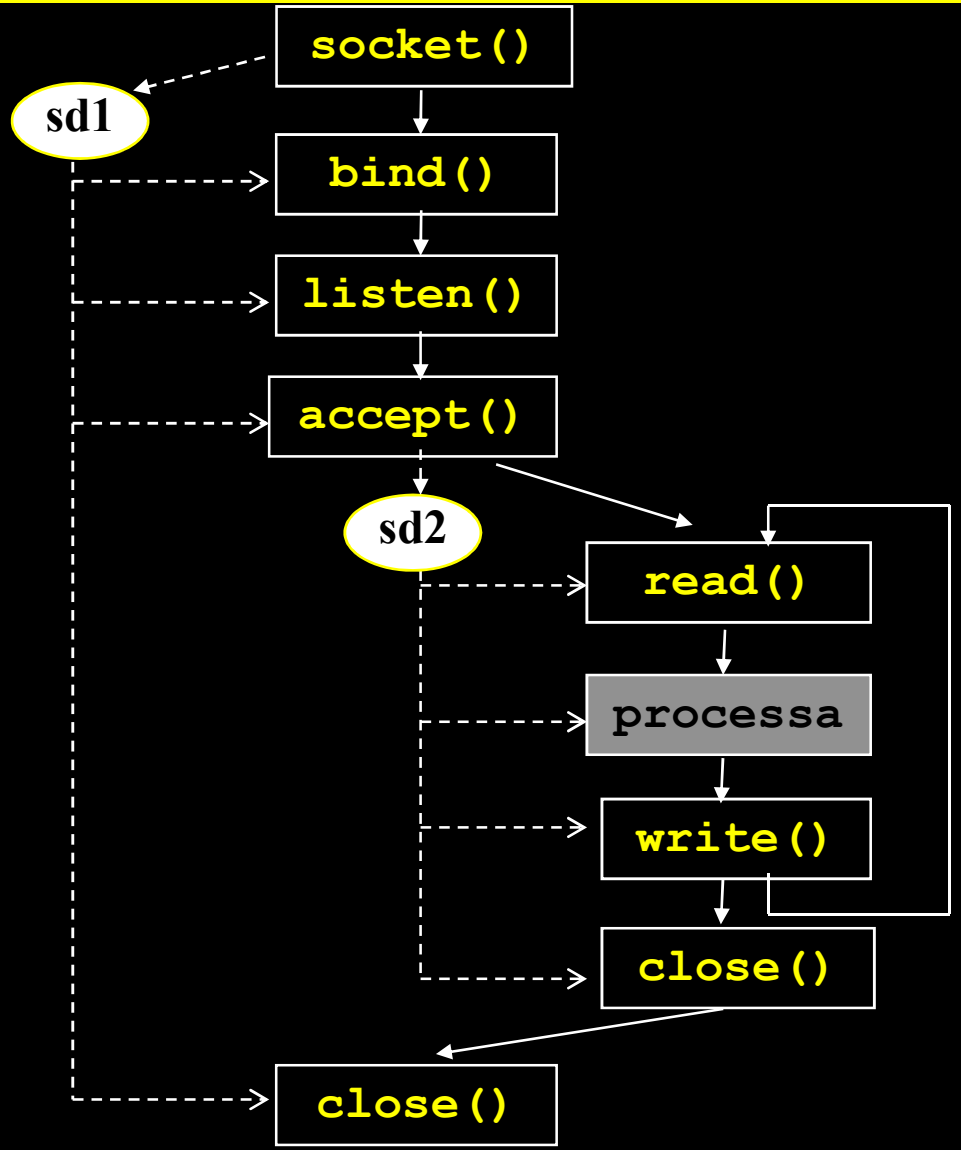


Exercício

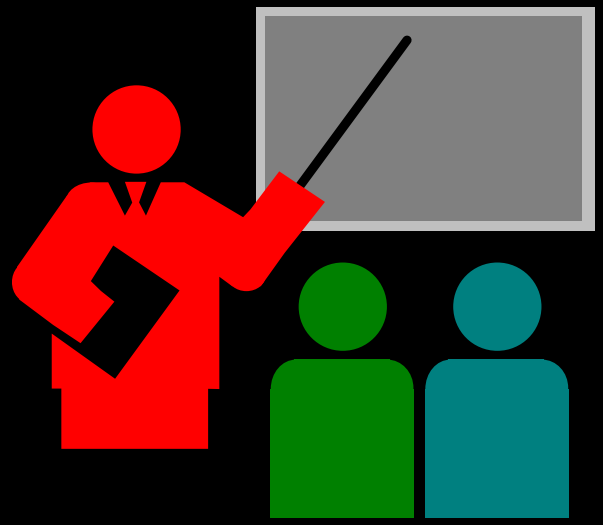
1. **Implemente um servidor TCP echo que transforma caracteres para maiúsculas.**
   - O programa deve atender a um cliente TCP e, quando receber a mensagem “quit” deve encerrar a conexão com o cliente e terminar o programa
   - Dicas:
     - Utilize a rotina de biblioteca `toupper()` para converter um caractere minúsculo para maiúsculo.
     - Utilize a rotina de biblioteca `strcmp()` para comparar a string recebida com “quit”.

```
#include <string.h>
If (strcmp(bufferp,“quit”)==0)
    /* igual */
```

Exercício


socket()
sd1
bind()
listen()
accept()
sd2
read()
processa
write()
close()
close()

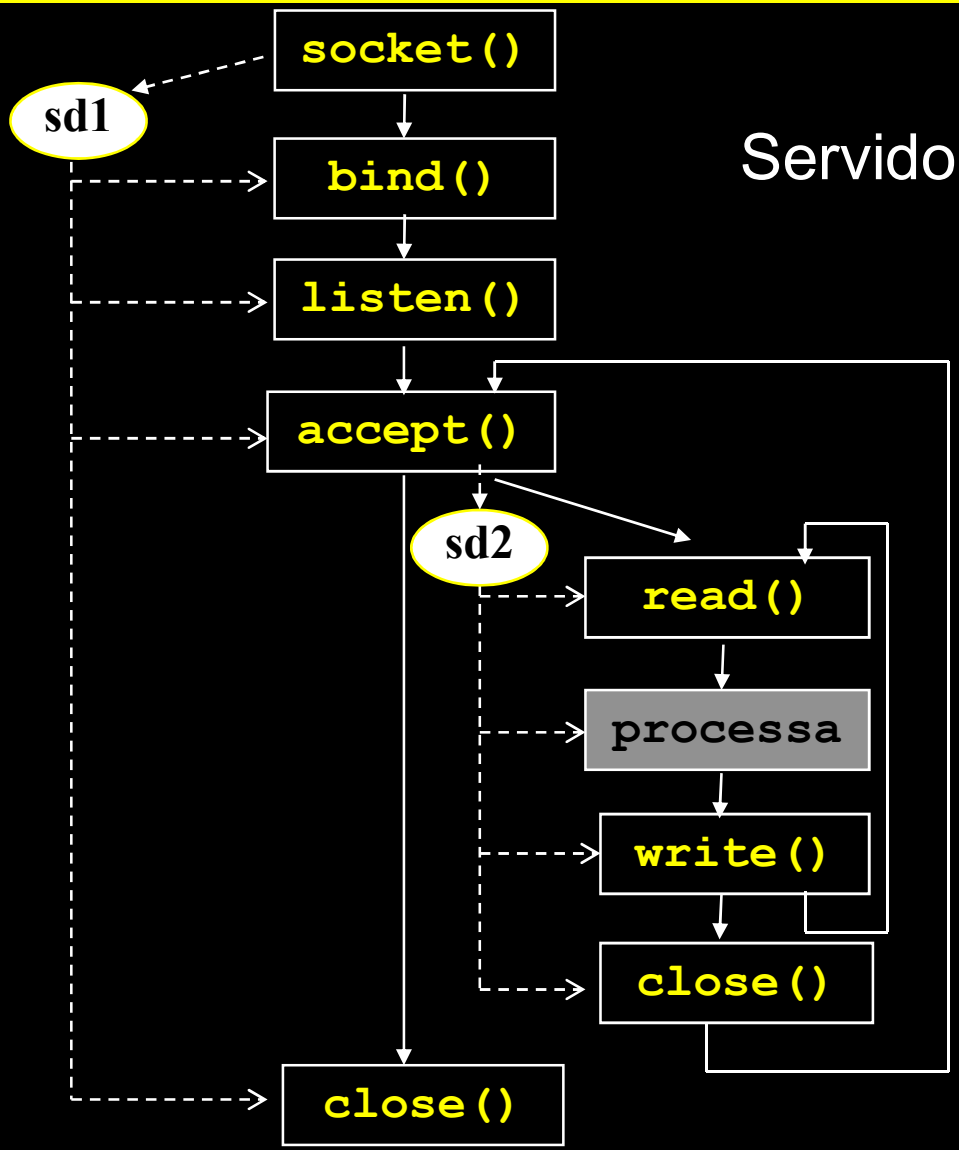
Servidor não concorrente
x
Servidor concorrente



Servidor não concorrente x concorrente

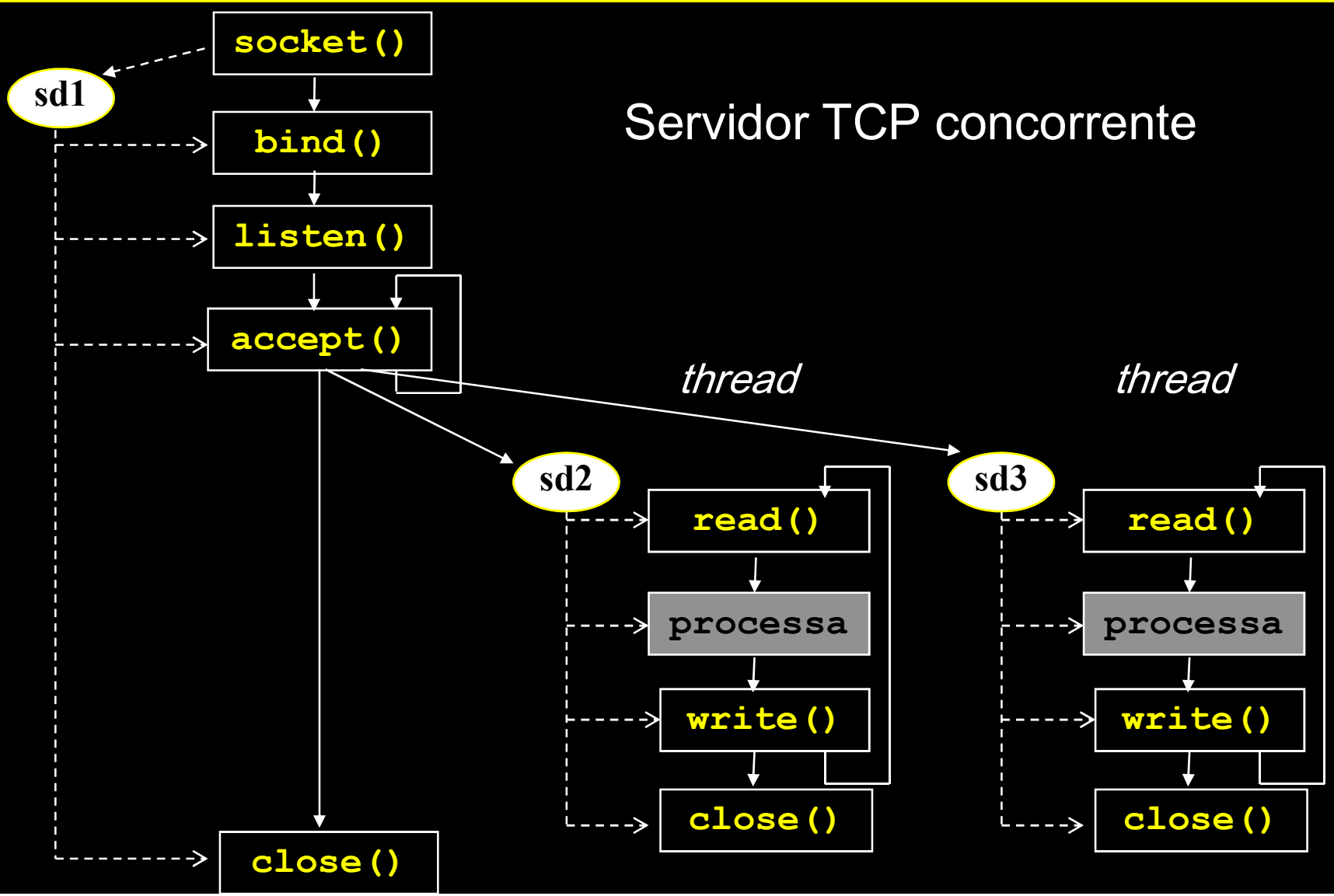
- **Servidor não concorrente**
  - Processa uma requisição por vez
- **Servidor concorrente**
  - Mais eficiente: tem capacidade para processar mais que uma requisição simultaneamente
  - Mais complexo
  - Mais difícil de implementar
  - Alternativas para implementação:
    - Utilização de chamadas assíncronas (muito complexo)
    - Servidor multithreaded

Servidor não concorrente x concorrente

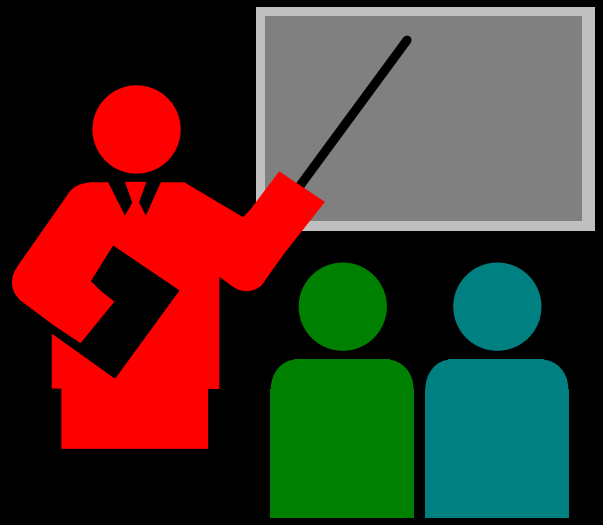


Servidor TCP não concorrente

Servidor não concorrente x concorrente



Exercício

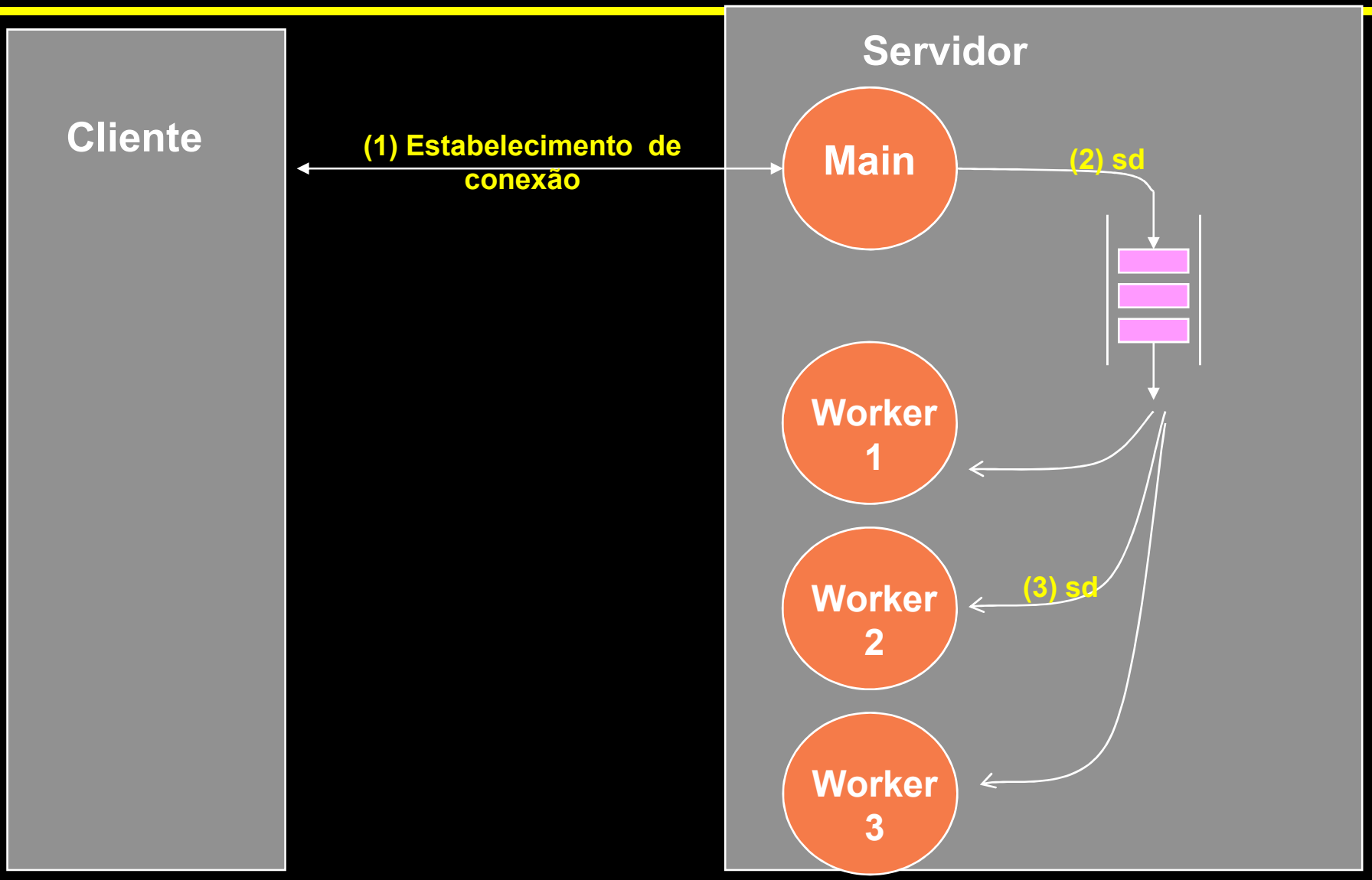


Exercício

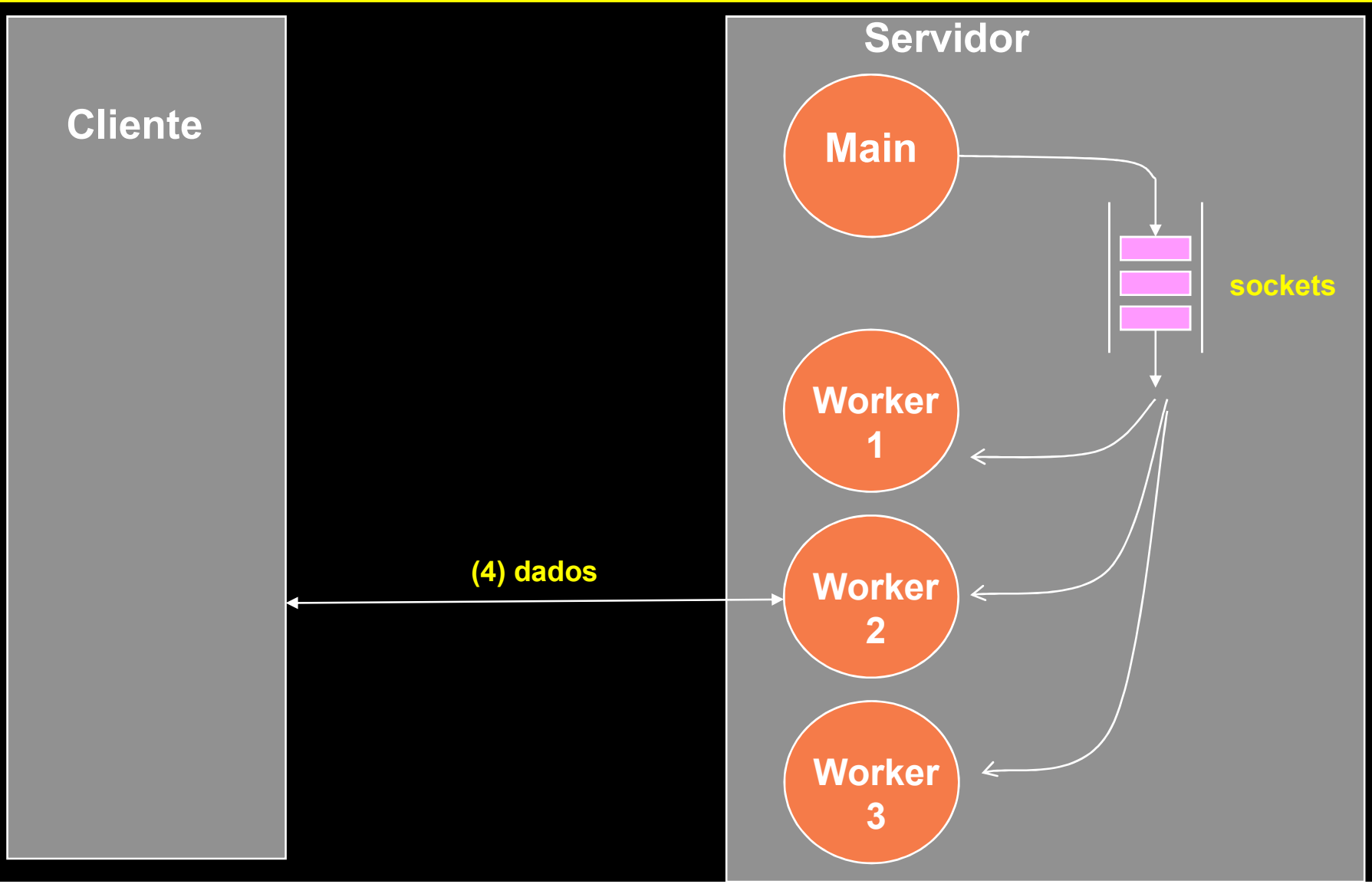
2. Fazer um servidor TCP echo concorrente.

- Dica: existem diversas formas de implementar o controle dos threads:
- (a) Utilizar uma tabela de controle de conexões estabelecidas com cada entrada da tabela contendo os seguintes campos: ocupado, estrutura dos threads (thread_t), semáforo para o thread e socket (new socket).
- (b) Utilizar a estrutura produtor-consumidor. O produtor é o thread principal: produz novas conexões (socket descriptors). O consumidor são os threads de trabalho: consomem conexões (socket descriptors). Um consumidor, ao receber o socket descriptor fica responsável pela interação com o cliente até que a conexão seja encerrada.

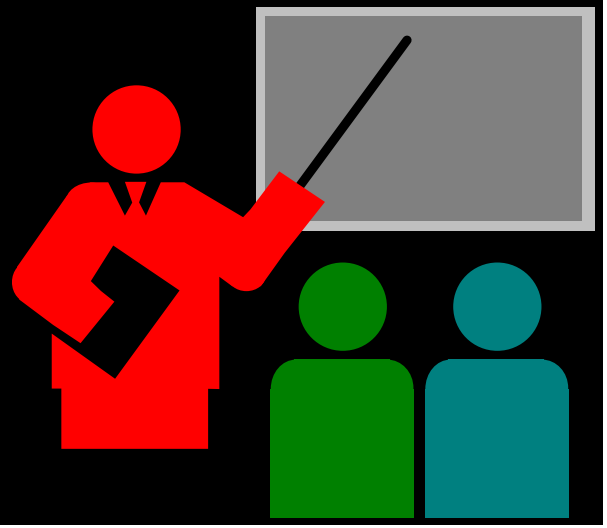
Exercício


Cliente
(1) Estabelecimento de conexão
Servidor
Main
(2) sd
Worker 1
Worker 2
(3) sd
Worker 3

Exercício


Cliente
Servidor
Main
sockets
Worker 1
Worker 2
Worker 3
(4) dados

Referências Bibliográficas



Referências Bibliográficas

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