Redes de Computadores
RCP 22108

Prof. Samir Bonho

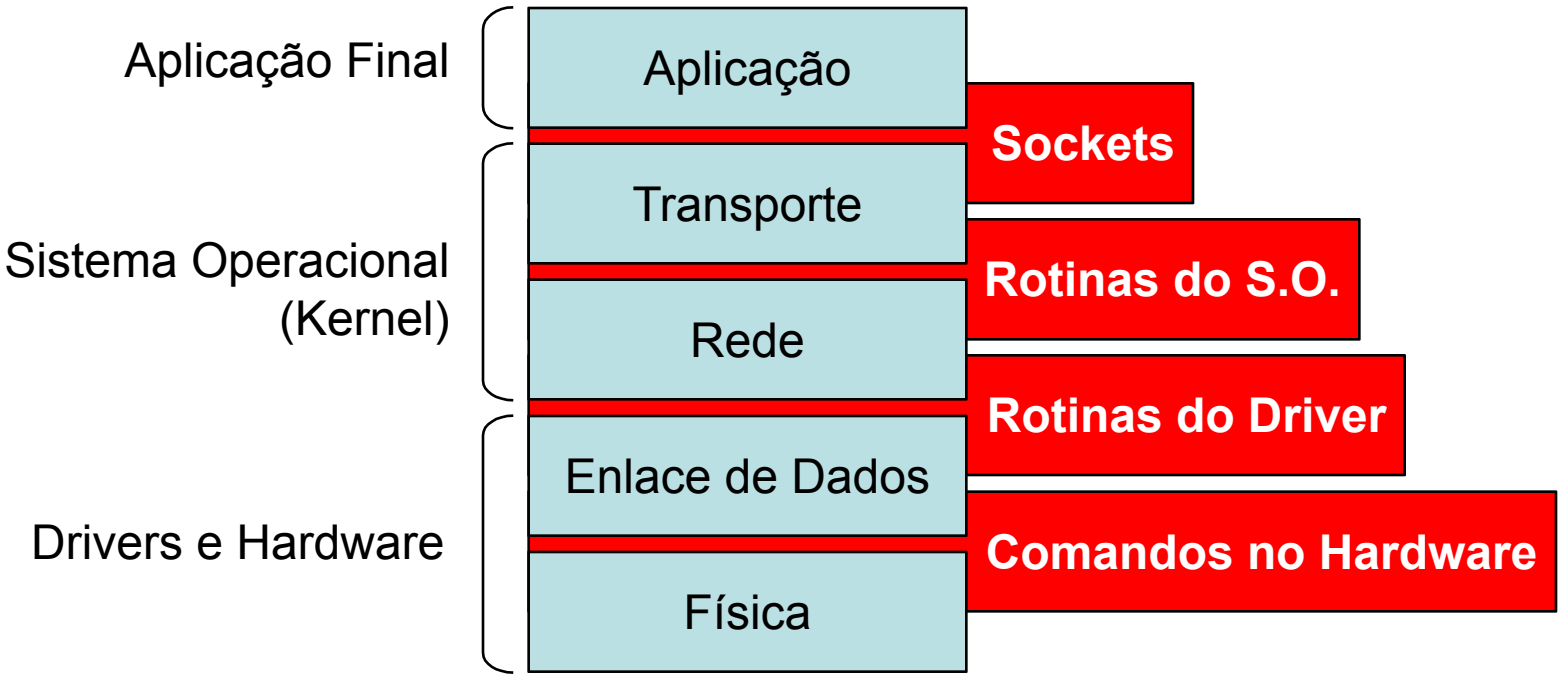
Engenharia Eletrônica

Sumário

- ✔ Sockets TCP

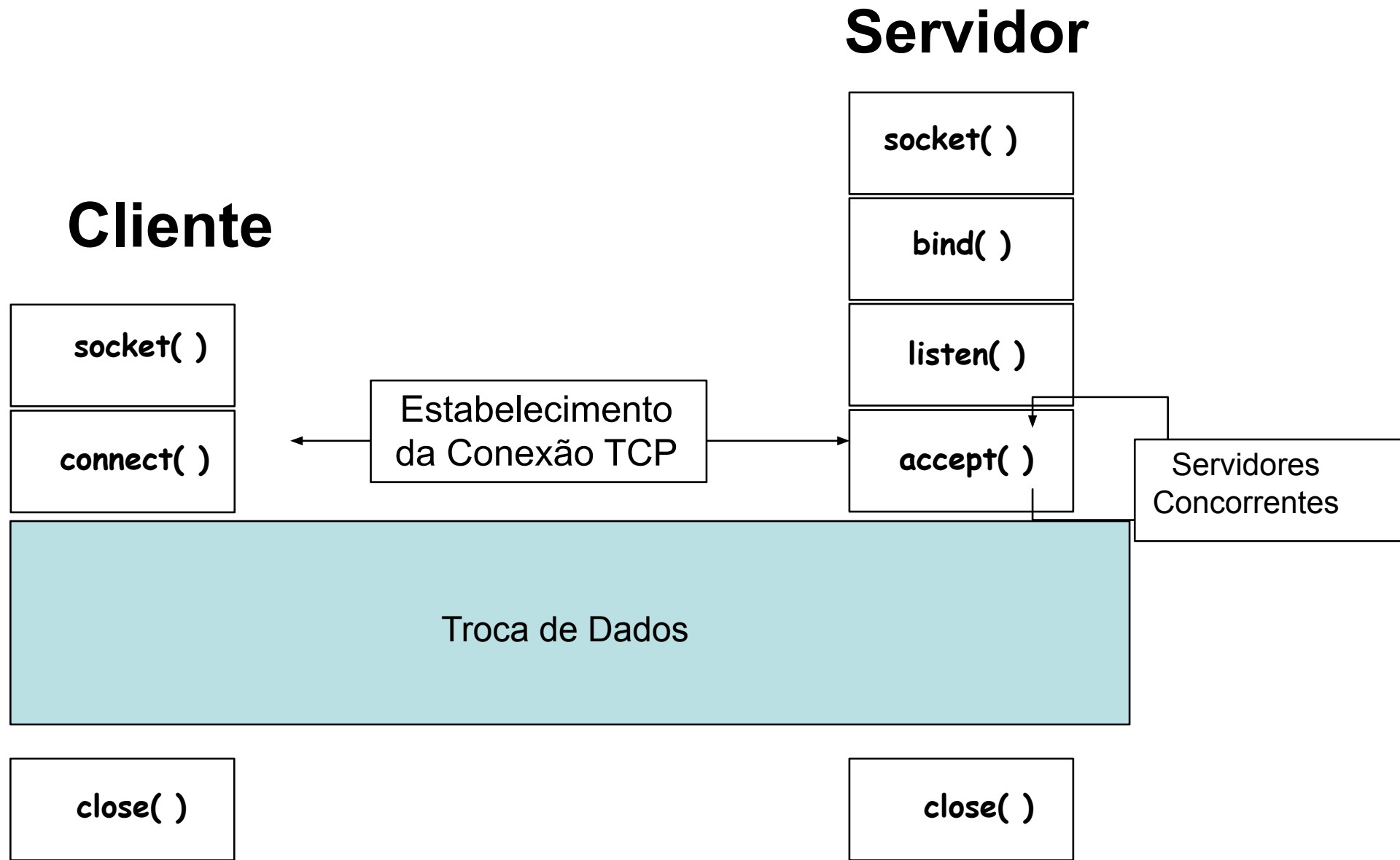
- ✔Fontes: Allan Lima – http://allanlima.wordpress.com
- ✔ Kurose

API Sockets


Aplicação Final
Sistema Operacional (Kernel)
Drivers e Hardware
Aplicação
Transporte
Rede
Enlace de Dados
Física
Sockets
Rotinas do S.O.
Rotinas do Driver
Comandos no Hardware

Tipos de Sockets

- Existem dois tipos básicos sockets:
  - Stream Sockets
  - Datagram Sockets
- **Stream Sockets geralmente utilizam o protocolo de comunicação TCP (*Transmission Control Protocol*)**
- Datagram Sockets geralmente utilizam o protocolo de comunicação UDP (*User Datagram Protocol*)


Servidor
socket( )
bind( )
listen( )
accept( )
Cliente
socket( )
connect( )
Estabelecimento
da Conexão TCP
Servidores
Concorrentes
Troca de Dados
close( )
close( )

Linux - Socket TCP usando C

Sockets - Criação

- Utiliza-se a chamada de sistema **socket()**:

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

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

- Parâmetros:
  - **domain** – o tipo de rede do socket. Usamos **AF_INET** (Address Family Internet). Outros tipos: **AF_LOCAL**, **AF_INET6**.
  - **type** – o tipo de serviço de transporte. Para sockets TCP usamos **SOCK_STREAM**, para UDP usamos **SOCK_DGRAM**.
  - **protocol** – o protocolo utilizado. Passando 0 (zero) é utilizado o padrão.
- A função retorna um descritor do socket criado ou –1 em caso de erro.

Sockets - Endereços

- Endereços IP e portas são armazenados em estruturas do tipo **struct sockaddr_in**.
- **sin_family** é o tipo do endereço. **AF_INET** deve ser usado para endereçamento IPv4.
- **sin_port** é a porta associada ao endereço.
- **sin_addr** é uma estrutura que contém o endereço IP (os 4 octetos).
- O endereço IP e a porta devem ser armazenados no byte order da rede (Big-Endian).

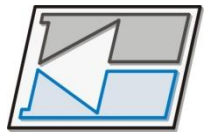
```
#include <netinet/in.h>

struct sockaddr_in {
    sa_family_t    sin_family;
    in_port_t      sin_port;
    struct in_addr sin_addr;
    /* ... outros campos */
};

struct in_addr {
    in_addr_t s_addr;
};

uint16 htons( uint16 data_in_host_order );
uint16 ntohs( uint16 data_in_net_order );

uint32 htonl( uint32 data_in_host_order );
uint32 ntohl( uint32 data_in_net_order );
```

Sockets - Associação

- Utiliza-se a chamada de sistema **bind()**:

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

int bind( int sockfd, struct sockaddr *my_addr, socklen_t addrlen );
```

- Parâmetros:
  - **sockfd** – o descritor do socket.
  - **my_addr** – a estrutura com o endereço para ser associado.
  - **addrlen** – o tamanho da estrutura do endereço.
- A função retorna 0 (zero) em caso de sucesso ou –1 no caso de um erro.
- Erro comum: EADDRINUSE (“Address already in use”)



Sockets – Traduzindo Endereços IP

- Para converter um endereço IP entre as formas de string e binária:

```
#include <arpa/inet.h>

int inet_aton(const char * str, struct in_addr * addrptr);
char * inet_ntoa(struct in_addr addr);
```

- A função **inet_aton()** converte um endereço na notação decimal (1.2.3.4) para o formato binário em byte order de rede (como a **struct sockaddr_in** espera) e retorna 0 (zero) em caso de sucesso.
- A função **inet_ntoa()** faz a conversão oposta.

Sockets – Resolvendo Nomes com DNS

- Para resolver nomes de hosts (www.ifsc.edu.br) para endereços IP (200.135.184.250) usamos as rotinas de acesso ao serviço DNS:

```
#include <netdb.h>

struct hostent * gethostbyname(const char * str);

struct hostent {
    int  h_length;          /* tamanho do endereço */
    char **h_addr_list;     /* lista de endereços */
    char *h_addr;           /* primeiro endereço */
    /* ... outros campos */
}
```

Sockets TCP

- Sockets orientados a conexão com garantias de entrega e ordenação.
- É preciso estabelecer a conexão antes da troca de dados.

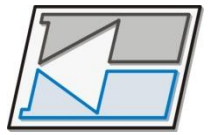
- Pedido de conexão recebido no socket de escuta (servidor), um novo socket é criado para realizar a comunicação.
- Assim é possível para o servidor voltar a aceitar novos pedidos de conexão mais tarde (usando o socket de escuta).

Sockets TCP – listen()

- Para por um socket em modo de escuta usamos a chamada de sistema **listen()**:

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

int listen( int sockfd, int backlog );
```

- Parâmetros:
  - **sockfd** – o descritor do socket.
  - **backlog** – a soma das filas de conexões completas e incompletas. Este parâmetro é extremamente dependente da implementação do Sistema Operacional.
- O valor de retorno da função é 0 (zero) em caso de sucesso ou –1 caso contrário.



Sockets TCP – accept()

- Esta função aceita pedidos de conexão pendentes ou fica bloqueada até a chegada de um.

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

int accept( int sockfd, struct sockaddr * cliaddr, socklen_t * addrlen
);
```

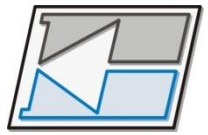
- Parâmetros:
  - **sockfd** – o descritor do socket.
  - **cliaddr** – a estrutura onde será guardado o endereço do cliente.
  - **addrlen** – argumento valor/resultado com o tamanho da estrutura do endereço.
- O **valor de retorno** da função é um **novo descritor** (não negativo) em caso de sucesso ou –1 caso contrário.
- Cada novo descritor retornado por **accept()** está associado à mesma porta do socket de escuta.

Sockets TCP – send()

- Usada para enviar dados por um socket conectado.

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

int send( int sockfd, void * buffer, size_t n_bytes, int flags );
```

- Parâmetros:
  - **sockfd** – o descritor do socket.
  - **buffer** – um ponteiro para os dados a serem enviados.
  - **n_bytes** – quantidade de bytes a serem enviados a partir do ponteiro **buffer**.
  - **flags** – opções para essa operação.
- O valor de retorno da função é a quantidade de bytes enviados em caso de sucesso ou –1 caso contrário.



Sockets TCP – recv()

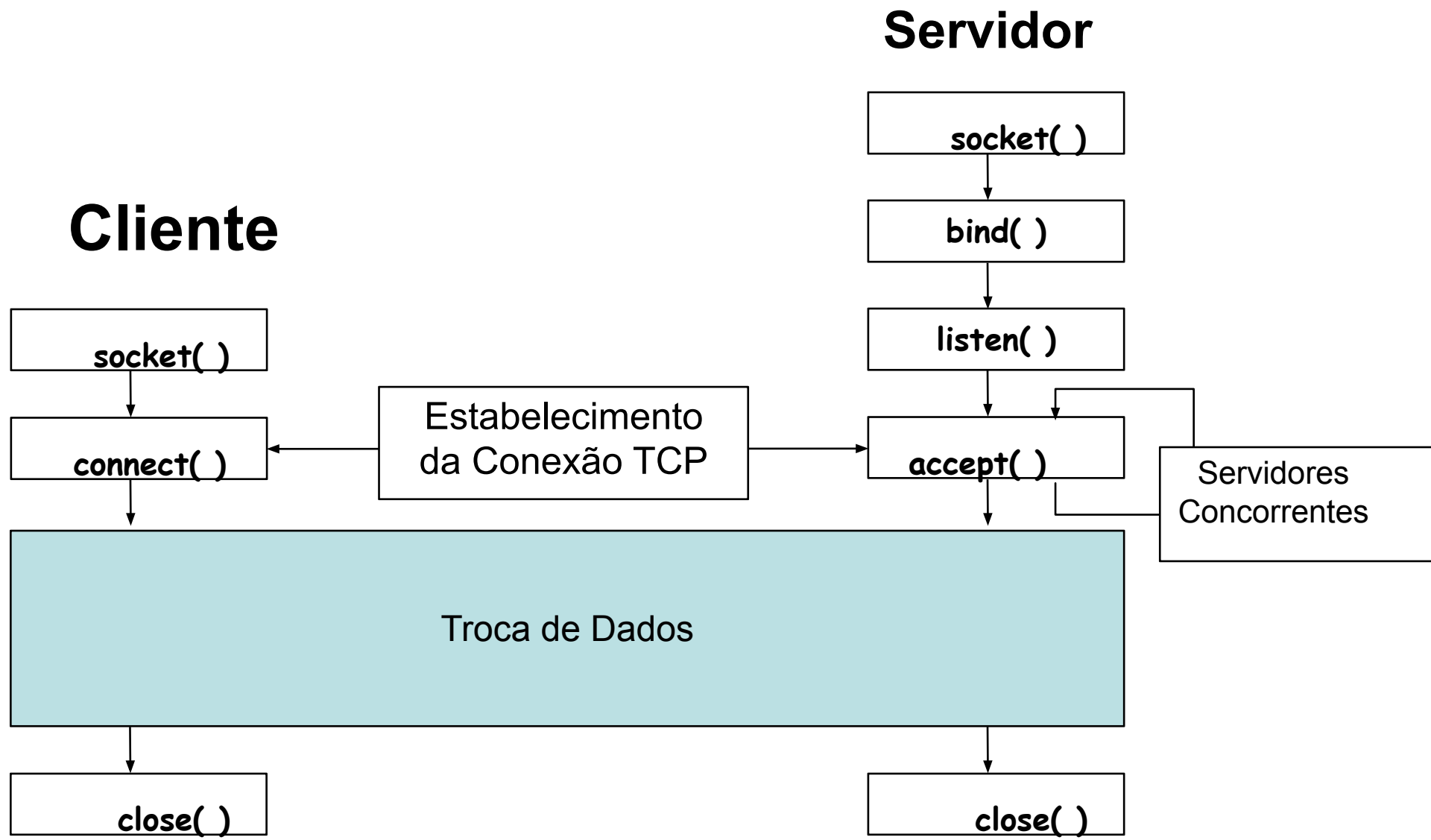
- Recebe dados por um descritor conectado, ou bloqueia a execução até que algum dado chegue ao socket:

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

int recv( int sockfd, void * buffer, size_t n_bytes, int flags );
```

- Parâmetros:
  - **sockfd** – o descritor do socket.
  - **buffer** – um ponteiro para a área de memória onde devem ser armazenados os dados recebidos.
  - **n_bytes** – quantidade máxima de bytes a serem recebidos.
  - **flags** – opções para essa operação.
- O valor de retorno da função é a quantidade de bytes recebidos em caso de sucesso ou –1 caso contrário.

Servidores Concorrentes

- Muitas vezes é necessário para um servidor lidar com vários clientes de uma única vez.
- Isto normalmente é feito através da criação de novas threads ou novos processos.
- Um servidor, após o retorno da função **accept()**, se divide em dois, e enquanto uma linha de execução se dedica a atender o cliente, outra volta a esperar por novos pedidos de conexão.


Servidor
socket( )
bind( )
listen( )
Cliente
socket( )
connect( )
Estabelecimento
da Conexão TCP
accept( )
Servidores
Concorrentes
Troca de Dados
close( )
close( )

Windows - Socket TCP usando C

Winsock

- É a API de sockets do windows
- Suporta diversos tipos de protocolos de transmissão
- Sua implementação é procedural
- Não é portável

Inicialização

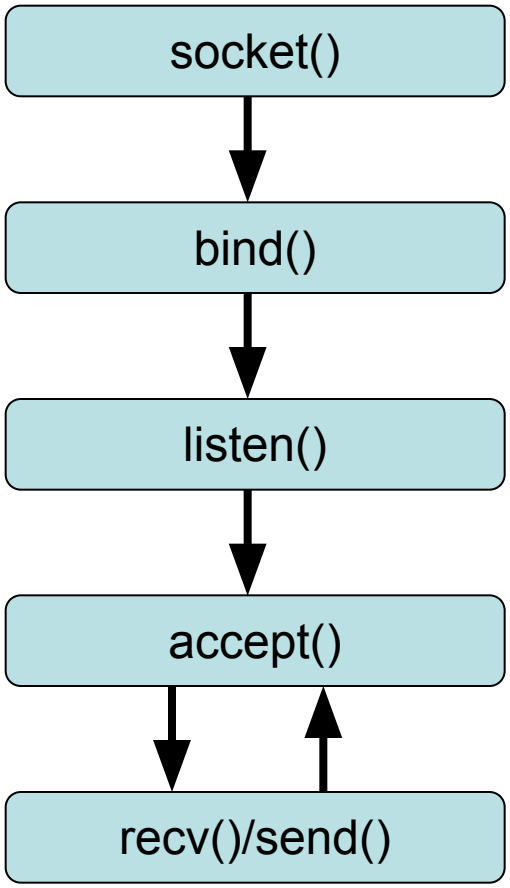
- Antes de usarmos as funções da Winsock devemos inicializa-la chamando a função WSAStartup

WSAStartup

```
int WSAStartup(
    WORD wVersionRequested,
    LPWSADATA lpWSAData
);
```

- *wVersionRequested* é versão que será usada
- *lpWSAData* é um ponteiro para a estrutura que irá guarda as informações sobre a implementação da versão usada
- Retorna zero em caso de sucesso ou um código de erro em caso de falha

Criando um servidor TCP



Cria o socket

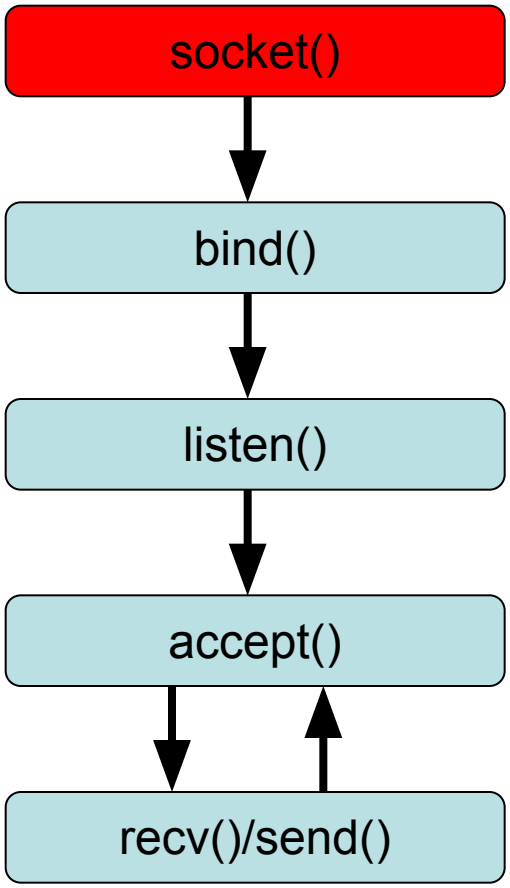
Associa o socket a uma porta

Configura a lista de conexões

Recebe uma conexão de um cliente

Envia e recebe dados

Criando um Socket



Cria o socket

Associa o socket a uma porta

Configura a lista de conexões

Recebe uma conexão de um cliente

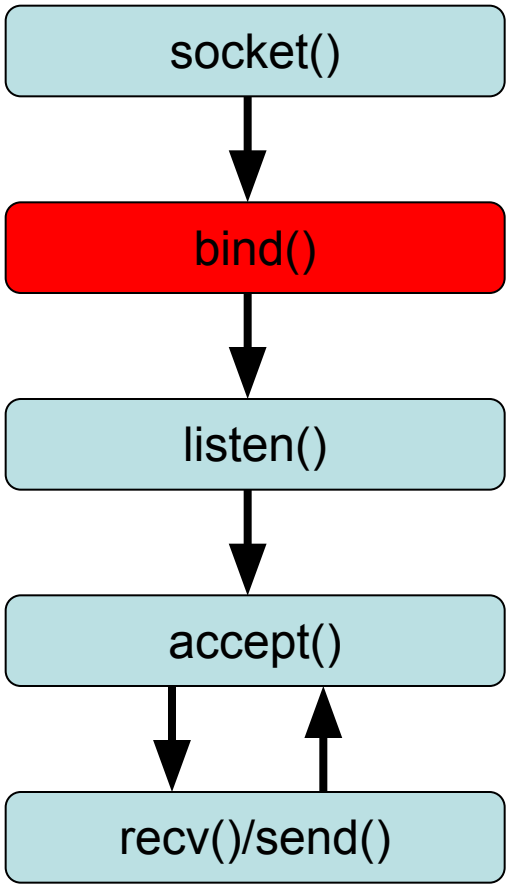
Envia e recebe dados

Criando um socket

```
SOCKET socket(
    int af, int type, int protocol
);
```

- *af* é a família do endereço
- *type* é o tipo do socket
  - Geralmente AF_INET
- *protocol* é o protocolo que será usado para a transmissão dos dados
  - IPPROTO_IP, IPPROTO_IPV6, IPPROTO_TCP, IPPROTO_UDP
- Retorna o descritor do socket

Associando uma porta



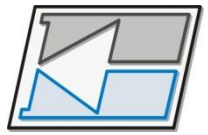
Cria o socket

Associa o socket a uma porta

Configura a lista de conexões

Recebe uma conexão de um cliente

Envia e recebe dados

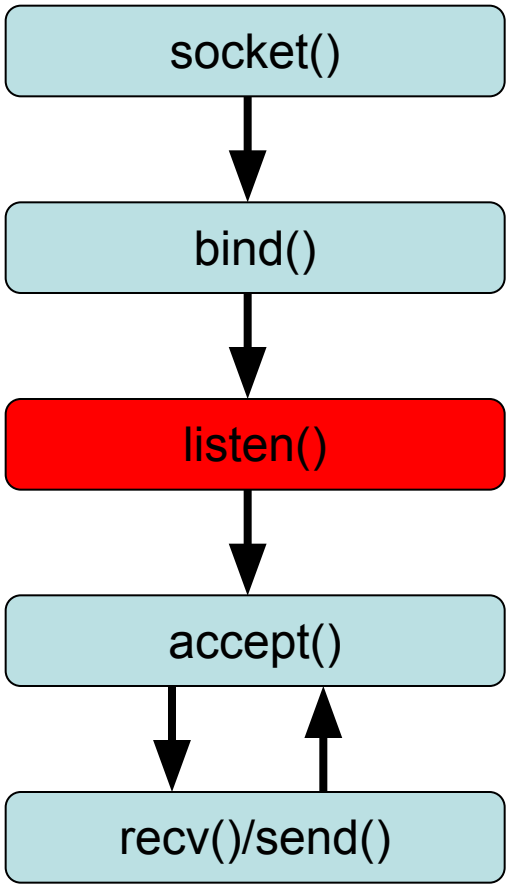


Associando uma porta

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

- s é o descritor do socket
- *name* contém as informações sobre o servidor
- *namelen* é o tamanho de *name* em bytes
- Retorna zero em caso se sucesso, quando falha retorna SOCKET_ERROR

Configurando a fila



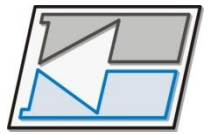
Cria o socket

Associa o socket a uma porta

Configura a lista de conexões

Recebe uma conexão de um cliente

Envia e recebe dados

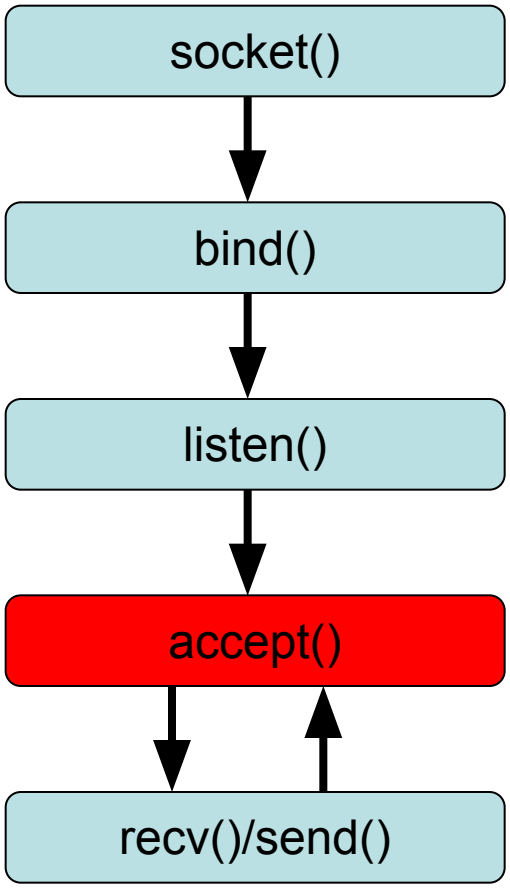


Configurando a fila

```
int listen(
    SOCKET s,
    int backlog
);
```

- s é o descritor do socket
- *backlog* é o tamanho máximo da fila de conexões
- Retorna zero em caso de sucesso e SOCKET_ERROR em caso de falha

Recebendo uma conexão



Cria o socket

Associa o socket a uma porta

Configura a lista de conexões

Recebe uma conexão de um cliente

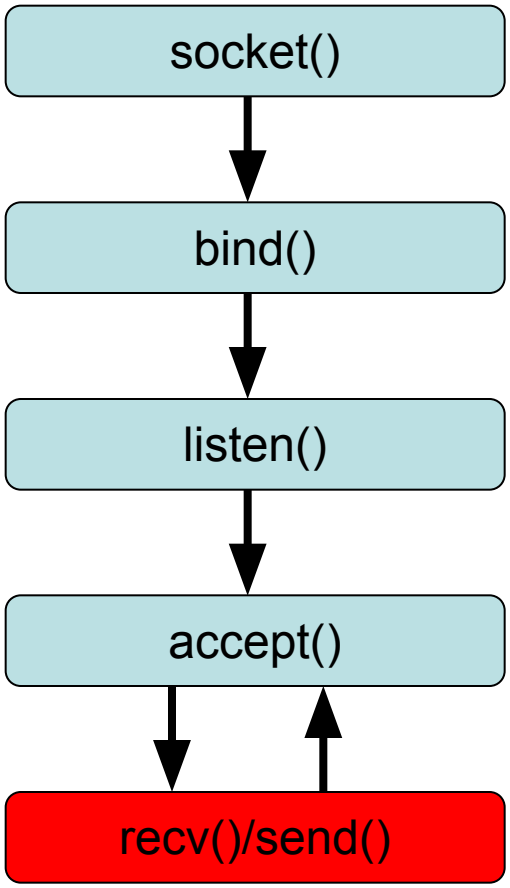
Envia e recebe dados

Recebendo uma conexão

```
SOCKET accept(
    SOCKET s,
    struct sockaddr* addr,
    int* addrlen
);
```

- *s* é o descritor do socket do servidor
- *addr* irá guardar as informações do cliente
- *addrlen* irá guardar o tamanho de *addr*
- Retorna o descritor do socket criado para o cliente

Enviando e recebendo dados



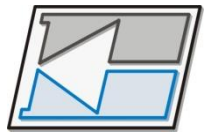
Cria o socket

Associa o socket a uma porta

Configura a lista de conexões

Recebe uma conexão de um cliente

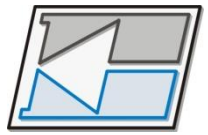
Envia e recebe dados



Recebendo dados

```
int recv(
    SOCKET s, char* buf,
    int len, int flags
);
```

- *s* é socket do qual se deseja receber os dados
- *buf* é o local onde os bytes lidos serão guardados
- len é o número máximo de bytes a serem lidos
- *flags* influenciam o comportamento da função
  - *MSG_PEEK,* MSG_OOB, MSG_WAITALL
- Retorna o número de bytes lidos em caso de sucesso ou SOCKET_ERROR em caso de falha



Enviando dados

```
int send(
    SOCKET s, const char* buf,
    int len, int flags
);
```

- *s* é socket pelo qual se deseja enviar os dados
- *buf* contém os dados a serem enviados
- len é o número máximo de bytes a serem enviados
- *flags* influenciam o comportamento da função
  - *MSG_DONTROUTE,* MSG_OOB
- Retorna o número de bytes envidados em caso de sucesso ou SOCKET_ERROR em caso de falha

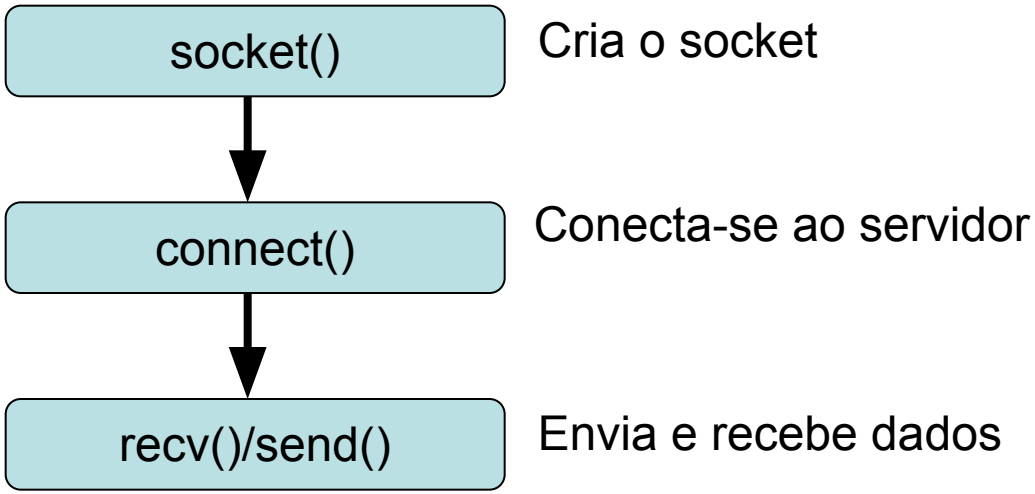
Exemplo Servidor TCP– API Windows

```
#include <winsock.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


int main(int argc, char *argv[])
{

	WSADATA wsInformacao;
	SOCKET socketServidor,socketCliente;
	struct sockaddr_in service, cliente;
	int iResult,tamamnhoEndereco,bytes;
	char bufferEntrada[101],bufferSaida[101];

	// inicializa Winsock
	iResult = WSAStartup(MAKEWORD(2,2), &wsInformacao);
	if (iResult != NO_ERROR) {
		printf("Erro na chamada da funcao WSAStartup().\n");
	}
```

```
socketServidor = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
   if (socketServidor == INVALID_SOCKET) {
         printf("Erro na chamada da funcao socket(): %ld.\n", WSAGetLastError());
         WSACleanup();
         return 0;
   }

   // asocia a uma porta o servidor


   service.sin_family = AF_INET;
   service.sin_addr.s_addr = inet_addr("127.0.0.1");
   service.sin_port = htons(27016);

   if (bind(socketServidor, (SOCKADDR *) &service, sizeof(service)) ==
SOCKET_ERROR ) {
         printf("Erro na chamada da funcao bind().\n");
         closesocket(socketServidor);
      return 0;
   }
```

```
    // configura a fila de conexões
    if (listen(socketServidor, 1) == SOCKET_ERROR) {
        printf("Erro na chamada da funcao listen().\n");
    }
    tamamnhoEndereco = sizeof(cliente);
    printf("Aguardando...\n");
    while (1) {
     socketCliente = SOCKET_ERROR;
        if ((socketCliente = accept(socketServidor, NULL, NULL)) ==
SOCKET_ERROR) {
            continue; }
        bytes = recv(socketCliente, bufferEntrada, 100, 0);
        bufferEntrada[bytes] = '\0';
        printf("Dados Recebidos: %s\n", bufferEntrada);

        sprintf(bufferSaida, "Ola %s!", bufferEntrada);
        send(socketCliente, bufferSaida, strlen(bufferSaida), 0);
        closesocket(socketCliente);
    }

    return 0;
}
```

Criando um cliente TCP

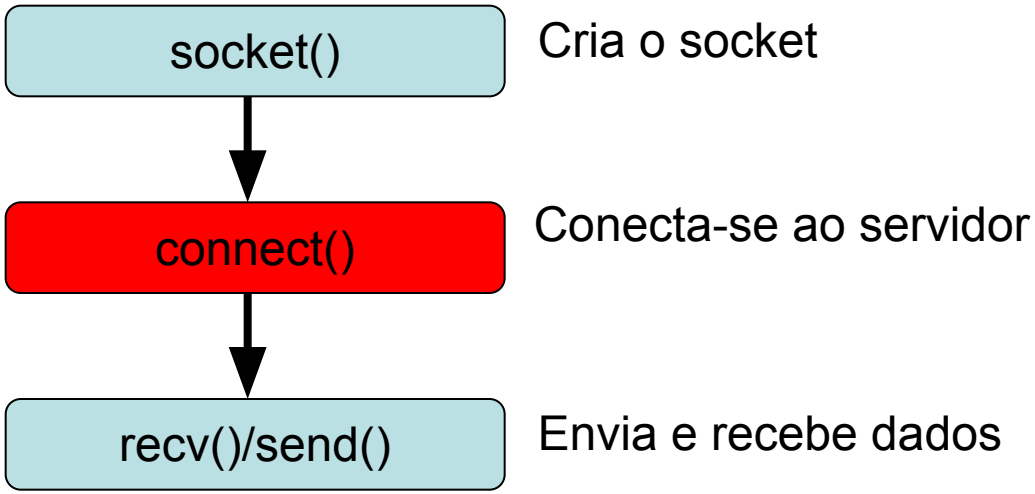


Cria o socket

Conecta-se ao servidor

Envia e recebe dados

Conectando-se ao servidor



Cria o socket

Conecta-se ao servidor

Envia e recebe dados

Conectando-se ao servidor

```
int connect(
    SOCKET s,
    const struct sockaddr* name,
    int namelen
);
```

- *s* é descritor do socket
- *name* guarda os dados do servidor com que a conexão será estabelecida
- *namelen* é o tamanho de *name* em bytes

Exemplo Cliente TCP– API Windows

```
#include <winsock.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{

    WSADATA wsInformacao;
    struct sockaddr_in servidor;
    int iResult,tamamnhoEndereco,bytes;
       SOCKET socketCliente;
       struct sockaddr_in cliente;
       char bufferEntrada[101];


       // inicializa Winsock
       iResult = WSAStartup(MAKEWORD(2,2), &wsInformacao);
       if (iResult != NO_ERROR) {
             printf("Erro na chadama da funcao WSAStartup()\n");
       }
```

```
socketCliente = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (socketCliente == INVALID_SOCKET) {
        printf("Erro na chadama da funcao socket(): %ld\n", WSAGetLastError());
        WSACleanup();
        return 0;
    }

    // se conecta ao servidor
    tamamnhoEndereco = sizeof(servidor);
    servidor.sin_family = AF_INET;
    servidor.sin_addr.s_addr = inet_addr("127.0.0.1");
    servidor.sin_port = htons(27016);

  if (connect(socketCliente, (SOCKADDR*) &servidor, sizeof(cliente)) ==
SOCKET_ERROR) {
      printf("Erro na chadama da funcao connect().\n");
      WSACleanup();
      return 0;
  }
```

```
    send(socketCliente, "Fulano", 6, 0);
        bytes = recv(socketCliente, bufferEntrada, 100, 0);
        bufferEntrada[bytes] = '\0';
        printf("Resposta: %s\n", bufferEntrada);


    closesocket(socketCliente);
    WSACleanup();

    system("PAUSE");
    return 0;
}
```

Resumo – Sockets TCP

- Servidor
  - Cria o *socket* servidor e aguarda conexão
  - Usa método accept() para pegar novas conexões
  - Cria streams entrada/saída para o *socket* da conexão
  - Faz a utilização dos *streams* conforme o protocolo
  - Fecha os *streams*
  - Fecha *socket* da conexão
  - Repete várias vezes
  - Fecha o *socket* servidor
- Cliente
  - Cria o socket com conexão cliente
  - Associa *streams* de leitura e escrita com o socket
  - Utiliza os *streams* conforme o protocolo do servidor
  - Fecha os *streams*
  - Fecha o *socket*

Resumo – Sockets TCP