Programação de Software Básico

Aula 9: Programação Cliente/Servidor

Apresentação

Você já se perguntou como os aplicativos da internet são construídos? A API do soquete e o suporte às comunicações TCP e UDP entre os hosts finais serão os tópicos principais desta aula, uma vez que a programação em soquete é a API principal utilizada para programar aplicativos distribuídos na Internet.

Para tanto, usaremos a linguagem de programação C. Na seção intitulada Explore +, você também encontrará referências para a API de soquetes nas linguagens Python e Java.

Objetivos

- Descrever os conceitos de soquete, modelo cliente-servidor e ordem de bytes;
- Examinar as funções da API de soquetes TCP e UDP;
- Compreender exemplos simples de aplicação cliente-servidor TCP e UDP.

Recordando conceitos básicos importantes

Clique nos botões para ver as informações.

Programa

É um arquivo executável armazenado no disco de um diretório. É lido na memória e executado pelo kernel como resultado da função exec(). Essa tem seis variantes, mas consideramos apenas a mais simples para esta aula.

Processo

É uma instância de execução de um programa. Às vezes, usa-se a nomenclatura tarefa em vez de processo, porém, o significado é mantido. O UNIX garante que todo processo tenha um identificador exclusivo chamado ID do processo. Esse é sempre um número inteiro não negativo.

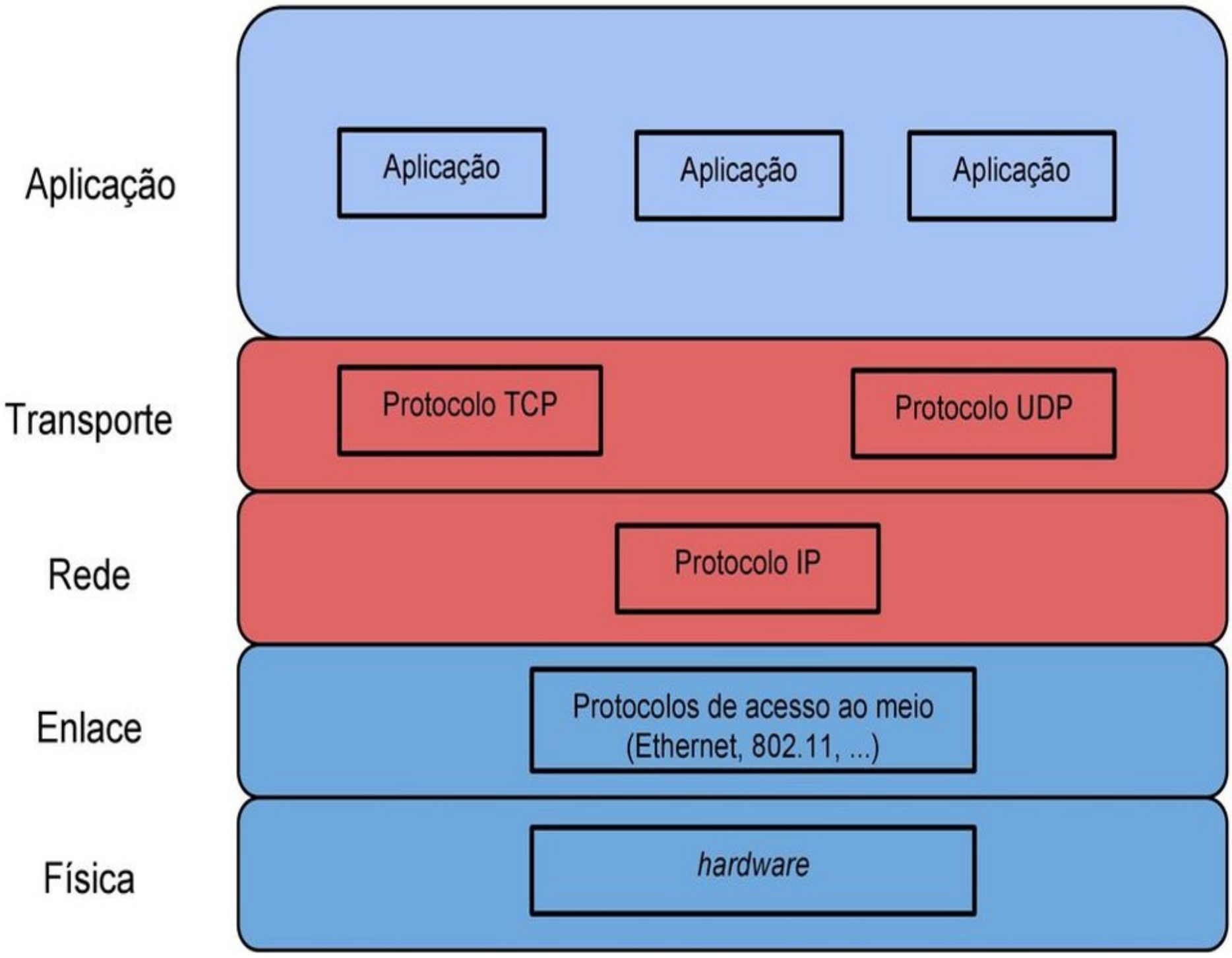
Descritores de arquivo

Normalmente, são números inteiros não negativos que o kernel usa para identificar os arquivos que estão sendo acessados por um processo específico. Sempre que se abre um arquivo existente ou se cria um, o kernel retorna um descritor de arquivo usado para ler ou gravar o arquivo. Os soquetes são baseados em um mecanismo muito semelhante, conforme você verá ao estudar os descritores de soquete.

O modelo cliente-servidor

Esse modelo é um dos paradigmas de comunicação mais utilizados em sistemas em rede. Os clientes, normalmente, se comunicam com um servidor por vez. Da perspectiva do servidor, não é incomum que possa se comunicar com vários clientes. Eles precisam saber da existência e do endereço do servidor, mas o contrário não acontece, visto que o servidor não precisa saber o endereço, ou mesmo a existência, do cliente antes da conexão ser estabelecida.

Cliente e servidores se comunicam por meio de várias camadas de protocolos de rede. Por enquanto, nosso foco será o conjunto de protocolos TCP / IP, conforme ilustrado na figura a seguir.



Aplicações (clientes e/ou servidoras), e a pilha de protocolos TCP/IP.(Fonte: Docplayer).

Protocolo de datagrama de usuário (UDP)

O UDP – *User Datagram Protocol* – é um protocolo simples da camada de transporte. O aplicativo grava uma mensagem em um soquete UDP, que é então encapsulado em um datagrama UDP, que, por sua vez, é encapsulado em um datagrama IP e enviado ao destino.

Não há garantia de que o UDP chegará ao destino ou que a ordem dos datagramas será preservada na rede ou, ainda, que os datagramas chegarão apenas uma vez. O problema do UDP é sua falta de confiabilidade: se um datagrama atinge seu destino, mas a soma de verificação detecta um erro ou se o datagrama é descartado na rede, ele não é retransmitido automaticamente.

Cada datagrama UDP é caracterizado por um comprimento, que é passado para o aplicativo de recebimento junto com os dados. Nenhuma conexão é estabelecida entre o cliente e o servidor e, por esse motivo, dizemos que o UDP fornece um serviço sem conexão.

O UDP está descrito na RFC 768.

Protocolo de controle de transmissão (TCP)

O TCP – *Transmission Control Protocol* – fornece um serviço orientado à conexão, pois é baseado em conexões entre clientes e servidores. Esse protocolo fornece confiabilidade. Quando um cliente TCP envia dados para o servidor, é necessário um reconhecimento em troca. Se uma confirmação não for recebida, o TCP retransmitirá automaticamente os dados e aguardará um período maior.

Mencionamos que os datagramas UDP são caracterizados por um comprimento. Em vez disso, o TCP é um protocolo de fluxo de bytes, sem nenhum limite.

Atenção

O TCP é descrito nas RFC 793, RFC 1323, RFC 2581 e RFC 3390.

Endereços de soquete

A estrutura de endereço do soquete IPv4 é denominada sockaddr_in e é definida pela inclusão do cabeçalho <netinet/in.h>. A definição POSIX é a seguinte:

```
struct in_addr{
in_addr_t s_addr; /*Endereço Ip de 32 bits, no byte order da rede */
};
struct sockaddr_in {
 uint8_t sin_len; /* length of structure (16)*/
 sa_family_t sin_family; /* AF_INET*/
 in_port_t sin_port; /* Porta, de 16 bits, TCP ou UDP*/
 struct in_addr sin_addr; /* Endereço IPv4 de 32 bits*/
 char sin_zero[8]; /* não é usado, mas deve ser inicializado com zeros*/
};
```

O tipo de dados uint8_t é um número inteiro de 8 bits sem sinal.

Estrutura de endereço de soquete genérico

Uma estrutura de endereço de soquete sempre é passada por referência como argumento para qualquer função de soquete. Mas qualquer função de soquete que use um desses ponteiros como argumento deve lidar com estruturas de endereço de soquete de qualquer uma das famílias de protocolos suportadas.

Surge um problema ao declarar o tipo de ponteiro que é passado. Com o ANSI C, a solução é usar o void * (o tipo genérico de ponteiro), porém, as funções de soquete antecedem a definição de ANSI C e a solução escolhida foi definir um endereço de soquete genérico da seguinte maneira:

```
struct sockaddr {
 uint8_t sa_len;
 sa_family_t sa_family; /* Família de endereços. Devemos usar AF_INET para comunicação via TCP/IP */
 char sa_data[14];
};
```

Ordem de bytes do host e conversão de ordem de bytes da rede

Existem duas maneiras de armazenar dois bytes na memória:

- com o byte de ordem inferior no endereço inicial − ordem dos bytes *little-endian*;
- com o byte de ordem superior no endereço inicial − ordem dos bytes da *big-endian*.

Nós os chamamos coletivamente de ordem de bytes de host.

Exemplo

Por exemplo, um processador Intel armazena o número inteiro de 32 bits como quatro bytes consecutivos na memória na ordem 1-2-3-4, em que 1 é o byte mais significativo (*big-endian*). Os processadores IBM PowerPC armazenariam o número inteiro na ordem de bytes 4-3-2-1, sendo 4 o byte menos significativo (*little-endian*).

Protocolos de rede, como o TCP, são baseados em uma ordem específica de bytes da rede. Os protocolos da internet usam pedidos de bytes big-endian.

As funções htons (), htonl (), ntohs () e ntohl ()

As seguintes funções são usadas para a conversão:

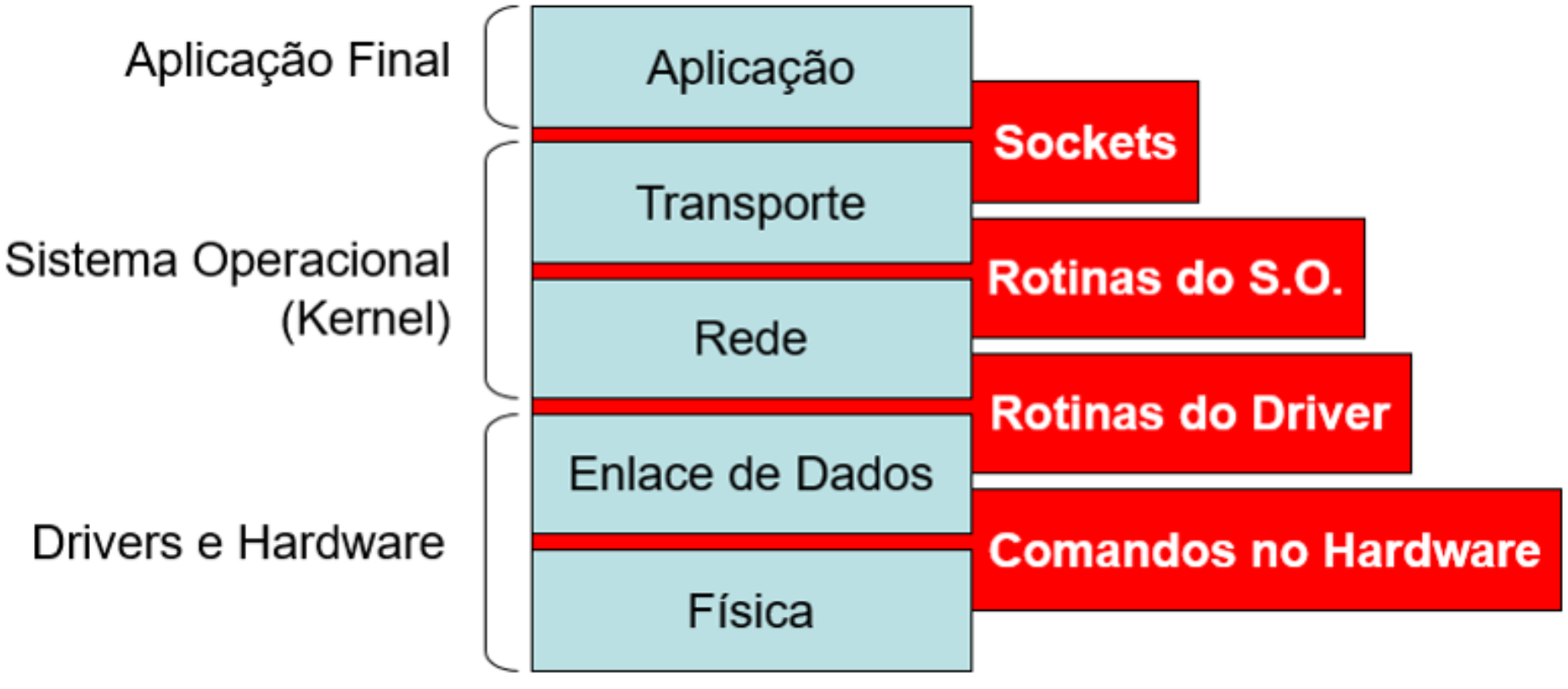
TÍTULO

```
#include <netinet/in.h>
uint16_t htons(uint16_t host16bitvalue);
uint32_t htonl(uint32_t host32bitvalue);
uint16_t ntohs(uint16_t net16bitvalue);
uint32_t ntohl(uint32_t net32bitvalue);
```

Os dois primeiros retornam o valor na ordem de bytes da rede (16 e 32 bits, respectivamente). Os últimos retornam o valor na ordem de bytes do host (16 e 32 bits, respectivamente).

API de soquetes: O que é?

A API de soquetes ou API de sockets é um elemento de ligação entre a aplicação e um sistema de mais baixo nível, em geral, o sistema operacional (S.O.), como você pode ver a seguir. Uma boa definição para isso é dizer que se trata de um conjunto de funções disponível aos programadores, em diversas linguagens de programação, para que estes implementem troca de dados entre processos remotos.



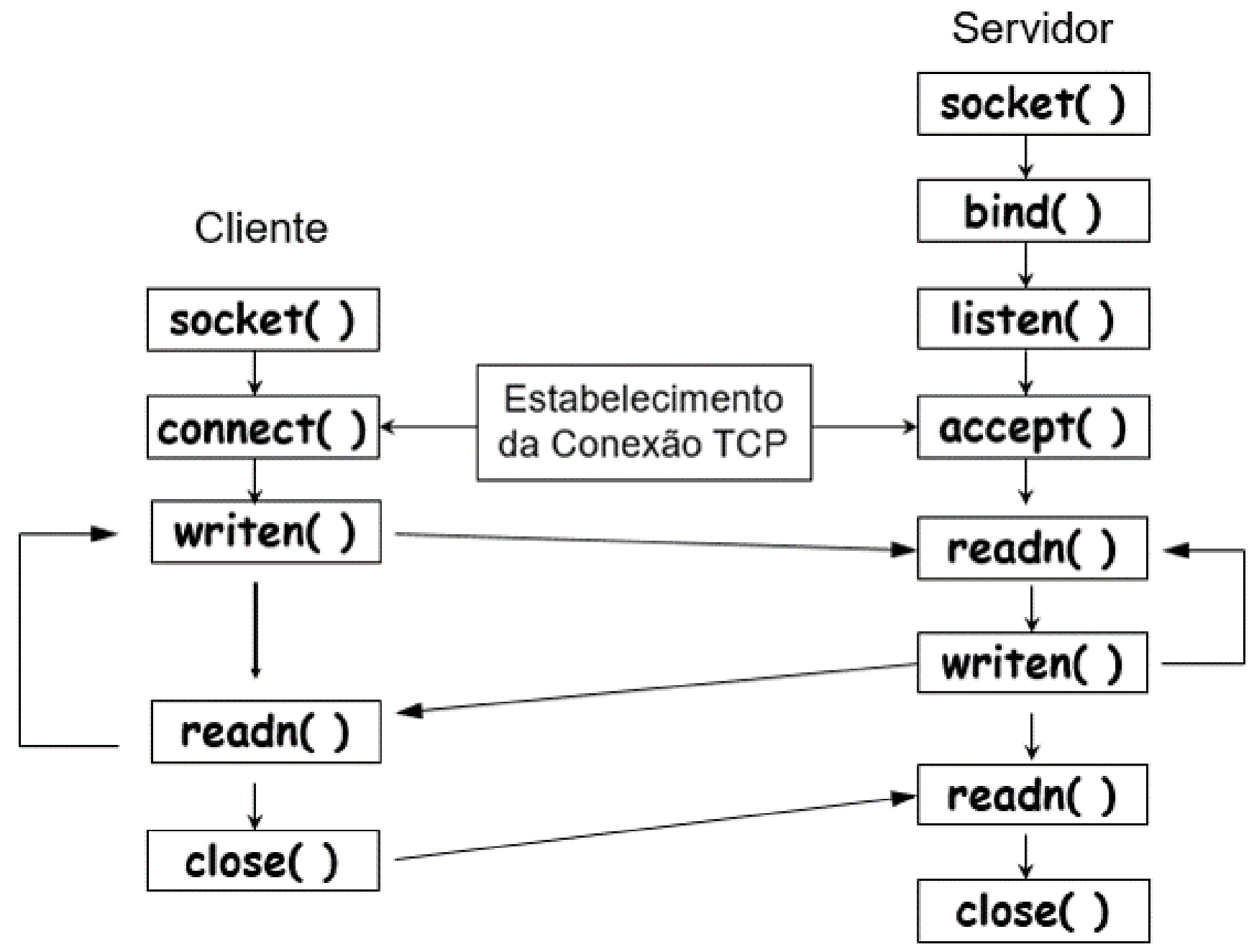
API de Sockets: Ligação entre a aplicação e o S.O. para implementação de troca remota de dados

A API de sockets pode ser dividida em dois subtipos:

API de sockets TCP	API de sockets UDP

API de soquete TCP

A sequência de chamadas de função para o cliente e um servidor participando de uma conexão TCP é apresentada na figura a seguir.



Cliente-Servidor TCP: Esquema típico de interação

Como pode ver, as etapas para estabelecer um soquete TCP no lado do cliente são as seguintes:

- Crie um soquete usando a função socket();
- Conecte o soquete ao endereço do servidor com a função connect();
- Envie e receba dados por meio das funções read() e write();
- Feche a conexão inserindo a função close().

As etapas envolvidas no estabelecimento de um soquete TCP no lado do servidor são as seguintes:

Crie um soquete com a função socket();

Vincule o soquete a um endereço usando a função bind();

Para conexões, insira a função listen();

Aceite uma conexão com a chamada do sistema da função accept(). Essa chamada normalmente é bloqueada até que um cliente se conecte ao servidor;

Envie e receba dados por meio de send() e receive();

Feche a conexão com a função close().

Funções

Clique no botão acima.

A função socket ()

O primeiro passo é chamar a função de soquete, especificando o tipo de protocolo de comunicação (TCP baseado em IPv4, TCP baseado em IPv6, UDP). A função é definida da seguinte maneira:

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

A função retorna 0 se conseguir estabelecer uma conexão, ou seja, **handshake** de três vias TCP bem-sucedido ou -1 em caso contrário.

O cliente não precisa chamar bind() na seção antes de chamar esta função: o kernel escolherá uma porta disponível e o IP de origem, se necessário for.

A função connect()

A função connect() é usada por um cliente TCP para estabelecer uma conexão com um servidor TCP. Ela é definida da seguinte maneira:

```
#include <sys/socket.h>
int connect (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

Aqui, sockfd é o descritor de soquete retornado pela função de soquete.

A função retorna 0 se conseguir estabelecer uma conexão, ou seja, **handshake** de três vias TCP bem-sucedido ou -1 em caso contrário.

O cliente não precisa chamar bind() na Seção antes de chamar esta função: o kernel escolherá uma porta disponível e o IP de origem, se necessário for.

A função bind()

O bind () atribui um endereço de protocolo local a um soquete. Com os protocolos da internet, o endereço é a combinação de um endereço IPv4 ou IPv6 (32 bits ou 128 bits) junto com um número de porta TCP de 16 bits.

A função é definida da seguinte maneira:

```
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

Agora, sockfd é o descritor de soquete, servaddr é um ponteiro para um endereço específico de protocolo e addrlen é o tamanho da estrutura de endereço.

Também bind() retorna 0 se for bem-sucedido e -1 em caso de erro.

Esse uso do endereço de soquete genérico sockaddr requer que todas as chamadas para essas funções lancem o ponteiro para a estrutura de endereços específica do protocolo. Por exemplo, para e estrutura de soquete IPv4:

```
struct sockaddr_in serv; /* Estrutura para endereço socket IPv4 */
bind(sockfd, (struct sockaddr*) &serv, sizeof(serv))
```

Um processo pode ligar um endereço IP específico ao seu soquete: para um cliente TCP, isso atribui o endereço IP de origem que será usado para datagramas IP enviados nos soquetes. Para um servidor TCP, isso restringe o soquete a receber conexões de clientes de entrada destinadas apenas a esse endereço IP.

Normalmente, um cliente TCP não vincula um endereço IP ao seu soquete. O kernel escolhe o soquete IP de origem que está conectado, com base na interface de saída usada. Se um servidor TCP não vincular um endereço IP ao seu soquete, o kernel usará o endereço IP de destino dos pacotes recebidos como o endereço de origem do servidor.

Além disso, bind() permite especificar o endereço IP, a porta, ambas ou nenhuma.

A tabela abaixo resume as combinações para IPv4.

IP Address	IP Port	Comportamento
INADDR_ANY	0	O *Kernel* escolhe o IP e a porta
INADDR_ANY	non zero	O *Kernel* escolhe o IP e o processo especifica a porta
Local IP address	0	O processo especifica o IP e o *Kernel* especifica a porta
Local IP address	non zero	O processo especifica o IP e a porta

Observe que o endereço do host local é 127.0.0.1. Por exemplo, se você quisesse executar seu echoServer (conforme veremos mais adiante) em sua máquina local, seu cliente se conectaria ao 127.0.0.1 com a porta adequada.

A função listen()

A função listen() converte um soquete não conectado em um soquete passivo, indicando que o kernel deve aceitar solicitações de conexão recebidas direcionadas a esse soquete. É definido da seguinte forma:

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

Nesse caso, sockfd é o descritor de soquete e backlog é o número máximo de conexões que o kernel deve enfileirar para esse soquete. O argumento da lista de pendências fornece uma dica ao sistema sobre o número de solicitações de conexão pendentes que ele deve enfileirar em nome do processo. Quando a fila estiver cheia, o sistema rejeitará solicitações de conexão adicionais. O valor da lista de pendências deve ser escolhido com base na carga esperada pelo servidor.

A função listen() retorna 0 se for bem-sucedida e -1 em caso de erro.

A função accept()

O accept() é usado para recuperar uma solicitação de conexão e convertê-la em uma solicitação. É definido da seguinte forma:

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

Temos sockfd como um novo descritor de arquivo conectado ao cliente que chamou connect() . Os argumentos cliaddr e addrlen são usados para retornar o endereço de protocolo do cliente. O novo descritor de soquete tem o mesmo tipo de soquete e família de endereços do soquete original.

O soquete original passado para accept() não está associado à conexão, mas permanece disponível para receber solicitações de conexão adicionais. O kernel cria um soquete conectado para cada conexão de cliente que é aceita.

Se não nos importamos com a identidade do cliente, podemos definir o cliaddr e addrlen como NULL. Caso contrário, antes de chamar a função de accept, o parâmetro cliaddr deve ser definido como um buffer grande o suficiente para armazenar o endereço e definir o interger apontado por addrlen para o tamanho do buffer.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

A função send()

Como um terminal de soquete é representado como um descritor de arquivo, pode-se usar a leitura e gravação para se comunicar com um soquete enquanto estiver conectado. No entanto, se queremos especificar opções, precisamos de outro conjunto de funções.

Por exemplo, send() é semelhante ao write(), mas permite especificar algumas opções. Observe o exemplo abaixo, em que send() é definido da seguinte maneira:

```
#include
ssize_t send(int sockfd, const void *buf, size_t nbytes, int flags);
```

Aqui, buf e nbytes têm significado igual ao da gravação. Os sinalizadores de argumento adicionais são usados para especificar como queremos que os dados sejam transmitidos. Não consideraremos as opções possíveis nesta aula. Vamos assumir que é igual a 0.

Se for bem-sucedido, a função retorna o número de bytes. Caso contrário, -1.

A função receive()

A função recv() é semelhante a read(), mas permite especificar algumas opções para controlar como os dados são recebidos. Não consideraremos as opções possíveis nesta aula. Vamos assumir que é igual a 0.

Assim, recv é definido da seguinte maneira:

```
#include
ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags);
```

A função retorna o comprimento da mensagem em bytes, 0 se nenhuma mensagem estiver disponível e o ponto tiver feito um desligamento ordenado ou -1 em caso de erro.

A função close()

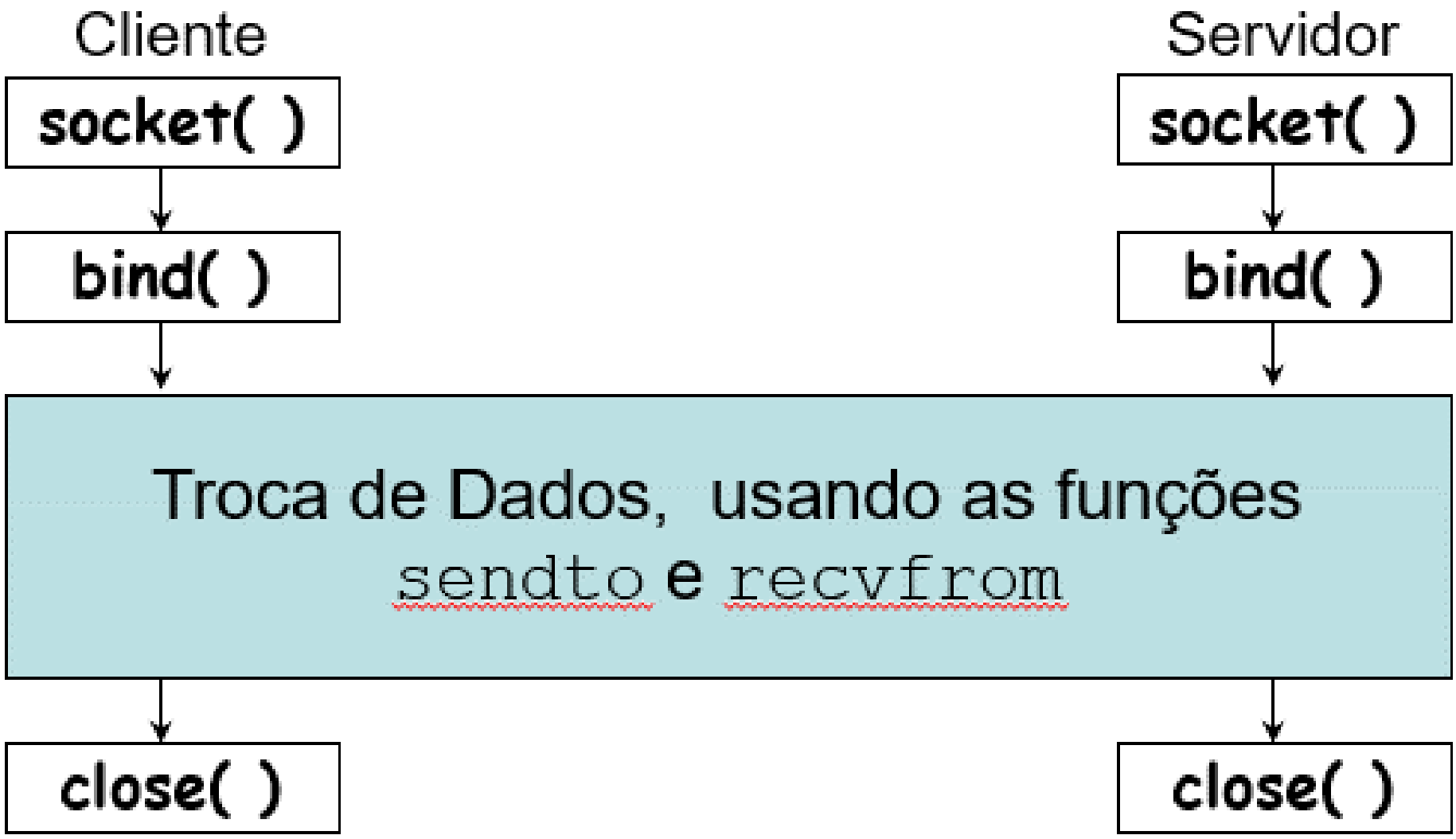
A função normal close() é usada para fechar um soquete e finalizar um soquete TCP. Retorna 0 se for bem-sucedido, -1 em caso de erro. É definido da seguinte forma:

```
#include
int close(int sockfd);
```

API de soquete UDP

Existem algumas diferenças fundamentais entre os soquetes TCP e UDP. O UDP é um protocolo de datagrama sem conexão, não confiável, enquanto o TCP é orientado à conexão, confiável e baseado em fluxo. Há alguns casos em que é usado o UDP em vez do TCP. Alguns aplicativos populares criados em torno do UDP são DNS, NFS, SNMP e, por exemplo, alguns serviços do Skype e mídia de streaming.

A figura mostra a interação entre um cliente e servidor UDP. Primeiramente, o cliente não estabelece uma conexão com o servidor. Em vez disso, o cliente apenas envia um datagrama ao servidor usando a função sendto , que requer o endereço do destino como parâmetro.



Cliente-Servidor UDP: Troca de dado usando as funções sento e recvfrom.

Da mesma forma, o servidor não aceita uma conexão de um cliente. Em vez disso, o servidor apenas chama a função recvfrom, que espera até que os dados cheguem de algum cliente. Depois, recvfrom retorna o endereço IP do cliente, juntamente com o datagrama, para que o servidor possa enviar uma resposta ao cliente. Da mesma forma, o servidor não aceita uma conexão de um cliente. Em vez disso, o servidor apenas chama a função recvfrom, que espera até que os dados cheguem de algum cliente. Depois, recvfrom retorna o endereço IP do cliente, juntamente com o datagrama, para que o servidor possa enviar uma resposta ao cliente.

Conforme mostrado, as etapas para estabelecer uma comunicação de soquete UDP no lado do cliente são as seguintes:

- Crie um soquete usando a função socket();
- Envie e receba dados por meio das funções recvfrom() e sendto().

As etapas para estabelecer uma comunicação de soquete UDP no lado do servidor são as seguintes:

- Crie um soquete com a função socket();
- Vincule o soquete a um endereço usando a função bind();
- Envie e receba dados por meio de recvfrom() e sendto().

Agora, iremos descrever as duas novas funções: recvfrom() e sendto().

Clique nos botões para ver as informações.

A função recvfrom ()

É semelhante à função *read()*, mas são necessários três argumentos adicionais. Essa função é definida da seguinte maneira:

```
#include
ssize_t recvfrom(int sockfd, void* buff, size_t nbytes,int flags, struct sockaddr* from,socklen_t *ad
drlen);
```

Os três primeiros argumentos, *sockfd* , *buff* e *nbytes*, são idênticos aos três primeiros argumentos de recv . Por sua vez, sockfd é o descritor de soquete, buff é o ponteiro para o qual escrever e nbytes é o número de bytes para escrever.
Em nossos exemplos, definiremos todos os valores do argumento *flags* como 0. O argumento *to* é uma estrutura de endereço de soquete que contém o endereço do protocolo, como por exemplo, o endereço IP e o número da porta de onde os dados são enviados. Por fim, *addlen* especificou o tamanho desse soquete.
A função retorna o número de bytes gravados, se for bem-sucedido, e -1 em caso de erro.

A função sendto ()

É semelhante à função *send*(), mas são necessários três argumentos adicionais. A função sendto() é definida da seguinte maneira:

```
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buff, size_t nbytes,int flags, const struct sockaddr *to,
socklen_t addrlen);
```

Os três primeiros argumentos, *sockfd* , *buff* e *nbytes*, são idênticos aos três primeiros argumentos de recv. Além disso, *sockfd* é o descritor de soquete, *buff* é o ponteiro para o qual escrever e *nbytes* é o número de bytes para escrever.
Em nossos exemplos, definiremos todos os valores do argumento *flags* como 0. O argumento *to* é uma estrutura de endereço de soquete que contém o endereço do protocolo, como por exemplo, endereço IP e número da porta de onde os dados são enviados. Por último, *addlen* especificou o tamanho desse soquete.
A função retorna o número de bytes gravados, se for bem-sucedido, e -1 em caso de erro.

A função connect() em sockets UDP

A princípio, pode parecer que não faz sentido o uso da função connect() em sockets UDP, já que se trata de um protocolo sem conexão. Entretanto, o uso da função connect() é possível em sockets UDP. Sua principal função é caracterizada pelo fato de o endereço de destino permanecer fixado, de forma que, após conectado, não serão mais passados o endereço do destino no 5º parâmetro das funções recvfrom() e sento(). Em vez disso, quando um socket UDP está conectado, passa-se NULL para o 5º parâmetro e 0 (zero) para 6º e último parâmetro.

Atenção

Note que, diferentemente do que ocorre em sockets TCP, o uso da chamada connect() em sockets UDP não gera um estabelecimento de conexão.

Exemplos de cliente / servidor TCP

Veja a seguir um exemplo completo da implementação de um servidor de eco baseado em TCP para resumir os conceitos apresentados. Trata-se de uma implementação iterativa e simultânea do servidor.

Exemplo de Cliente TCP: Conecta-se ao servidor, recebe uma string de 10 caracteres e a imprime na tela.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
int main( int argc, char ** argv )
{
 int sockfd;
 int sockfd;
 char recvline[10];
 struct sockaddr_in servaddr;
 if( argc != 2 )
 return -1;
 sockfd = socket( AF_INET, SOCK_STREAM, 0 );
    memset( &servaddr, 0,sizeof(servaddr) );
 servaddr.sin_family = AF_INET;
 servaddr.sin_port = htons(12345);
 inet_aton( argv[1],&servaddr.sin_addr );
 connect( sockfd,(struct sockaddr *)&servaddr, sizeof(servaddr) );
 recv( sockfd, recvline, 10,MSG_WAITALL );
 fputs( recvline, stdout );
 close( sockfd );

 return 0;
}
```

Exemplo de um servidor simples TCP: Aceita conexão e envia a string “Alo Mundo!” ao cliente.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
int main( int argc, char ** argv )
{
 int listenfd, connfd, size;
 struct sockaddr_in myaddr, cliaddr;
 listenfd = socket( AF_INET,SOCK_STREAM, 0 );
 memset( &myaddr, 0, sizeof(myaddr) );
 myaddr.sin_family = AF_INET;
 myaddr.sin_port = htons(12345);
 myaddr.sin_addr.s_addr =
 INADDR_ANY;
bind( listenfd, (struct sockaddr *)&myaddr, sizeof(myaddr) );
 listen( listenfd, 5 );
 for( ; ; )
 {
 memset( &cliaddr, 0,
 sizeof(cliaddr) );
 size = sizeof( cliaddr );
 connfd = accept( listenfd, (struct sockaddr *)&cliaddr, &size );
 send( connfd, “Alo Mundo!”, 10,MSG_WAITALL );
 close( connfd );
 }
 return 0;
}
```

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atividade

1. Qual é a função da chamada connect quando se trata de sockets TCP?

a) Apenas estabelecer conexão com o servidor.
b) Fixar o endereço de destino sem estabelecer conexão.
c) Fixar o endereço de destino e estabelecer uma conexão.
d) Receber dados pelo *socket* conectado.
e) Enviar dados pelo *socket* conectado.

2. Qual é a função da chamada connect quando se trata de sockets UDP?

a) Apenas estabelecer conexão com o servidor.
b) Fixar o endereço de destino sem o estabelecimento de conexão.
c) Fixar o endereço de destino e estabelecer uma conexão.
d) Receber dados pelo socket conectado.
e) Enviar dados pelo socket conectado.

---

3. Quanto às garantias oferecidas pela camada de transporte (S.O.) para o caso de sockets UDP, marque a opção correta:

a) Serão oferecidas garantias quanto às perdas.
b) Serão oferecidas garantias quanto à ordenação.
c) Serão oferecidas garantias quanto aos erros de transmissão (troca de bits).
d) Serão oferecidas garantias quanto à taxa mínima de transmissão (largura de banda).
e) Não será fornecida nenhuma garantia.

---

4. Quanto às garantias oferecidas pela camada de transporte (S.O.) para o caso de sockets TCP, marque a opção correta:

a) Serão oferecidas garantias quanto às perdas.
b) Serão oferecidas garantias quanto à ordenação.
c) Serão oferecidas garantias quanto aos erros de transmissão (troca de bits).
d) Serão oferecidas garantias quanto à entrega livre de perdas e de erros de transmissão, além da ordenação.
e) Não será fornecida nenhuma garantia.

---

5. Entre as funções da API socket estudadas nesta aula, temos uma responsável por associar um socket a uma porta de comunicação TCP/UDP. Marque a opção abaixo que indique corretamente a função que possui esta finalidade:

a) socket()
b) connect()
c) bind()
d) sendto()
e) recvfrom()

Notas

Sockets

O termo **socket** é traduzido por alguns autores para **soquete**. Por isso, usaremos as duas palavras para que você possa se acostumar com ambas.

Título modal [1]

Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos. Lorem Ipsum é simplesmente uma simulação de texto da indústria tipográfica e de impressos.

Referências

TANENBAUM, A. S.; BOS, H. **Sistema Operacionais Modernos**, Cap. 1, 3ª ed.

STEVENS, W. R; FENNER, B.; RUDOFF, A. M. **Programação de Rede Unix**. 3ª ed. Porto Alegre: Bookman, 2008.

Próxima aula

- Classes principais de servidores: iterativo e simultâneo;
- Técnica simples para servidor simultâneo: função fork e processo filho;
- Técnica alternativa: uso de threads (processos leves).

Explore mais

Assista ao vídeo:

- Tutorial de Programação de Sockets em C
- https://www.youtube.com/watch?v=LtXEMwSG5-8