Programação de Sockets em C/C++

Hernani Costa

hpcosta@dei.uc.pt

Redes de Comunicação 2011/2012

Introdução

- Os sockets são um método de criar conexão entre processos, independente do protocolo.

- Os sockets podem ser:
  - orientado à conexão ou não orientada à conexão - É estabelecida uma conexão antes da comunicação ou cada pacote descreve o destino?
  - orientado ao pacote ou orientado ao fluxo - Existe limite de mensagens ou é um fluxo (stream)?

  - Fiável ou Não Fiável - As mensagens podem ser perdidas, duplicadas, reordenadas ou corrompidas?

Caraterísticas de um Socket

- Os sockets são caraterizados pelo seu domínio, tipo e protocolo de transporte.

- Domínios comuns são:
  - AF_UNIX: o formato do endereço é o caminho UNIX
  - AF_INET: o formato do endereço é o *host* e o número da porta

Caraterísticas de um Socket

- Tipos comuns são:
  - Circuitos virtuais: Recebido pela ordem transmitida, fiável
  - Datagramas: Ordem arbitrária, não fiável

Caraterísticas de um Socket

- Cada tipo de socket tem um ou mais protocolos. Por exemplo:
  - TCP/IP (circuitos virtuais)
  - UDP (datagramas)

Caraterísticas de um Socket

- Os sockets orientados à conexão são utilizados em comunicações cliente-servidor: o servidor espera por uma conexão do cliente
- Os sockets não orientados à conexão são utilizados em sistemas peer-to-peer: cada processo é simétrico

API de Sockets

- socket - cria um socket de um domínio dado, tipo e protocolo (por exemplo, como comprar um telefone)
- bind - associa um nome ao socket (por exemplo, número de telefone)
- listen - especifica o número de conexões que podem estar em espera num socket de um servidor (por exemplo, chamadas telefónicas em espera)
- accept - o servidor aceita um pedido de conexão de um cliente (por exemplo, atender o telefone)

API de Sockets

- connect - o cliente faz o pedido de conexão ao servidor (por exemplo, fazer uma chamada)
- send, sendto - escrever para a conexão (por exemplo, falar ao telefone)
- recv, recvfrom - ler da conexão (por exemplo, ouvir ao telefone)
- shutdown - desligar a conexão

Comunicação orientada à conexão

O servidor executa as seguintes ações:

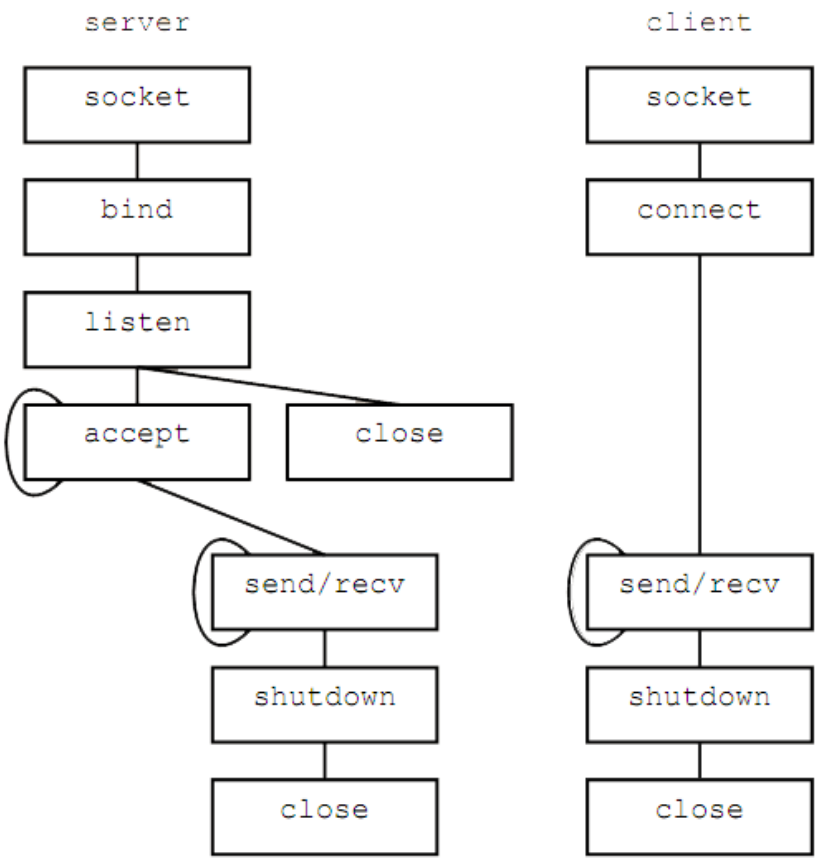
- socket - cria o socket
- bind - dá o endereço do socket no servidor
- listen - específica o número máximo de pedidos de conexões que podem estar em espera para este processo
- accept - estabelece conexão com o cliente especificado
- send, recv - equivalente ao à escrita e leitura, mas baseado no fluxo
- shutdown - fim da escrita ou leitura
- close - liberta as estrutura de dados do kernel

Cliente TCP

O cliente executa as seguintes ações:

- socket - cria o socket
- connect - conecta ao servidor
- send, recv - (repetido)
- shutdown
- close

Sockets TCP


server
socket
bind
listen
accept
close
send/recv
shutdown
close
client
socket
connect
send/recv
shutdown
close

Socket API

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


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

- Retorna um descritor (chamado socket ID) se criado com sucesso, -1 caso contrário
- O domain (domínio) é AF_INET

Socket API

- O type (tipo) pode ser:
  - SOCK_STREAM - estabelece um circuito virtual para o fluxo
  - SOCK_DGRAM - estabelece um datagrama para a comunicação
  - SOCK_SEQPACKET - estabelece uma conexão fiável, em dois sentidos com um tamanho de mensagem máximo (não está disponível na maioria das máquinas)
- O protocol (protocolo) geralmente é zero, sendo o type e o domain a definir a conexão

bind

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


int bind (int sid, struct sockaddr *addrPtr, int len);
```

- sid - é o ID do socket
- addrPtr - estrutura com o endereço (IP) e porta da máquina
- sid - é o tamanho (em bytes) da estrutura *addrPtr
- Associa um socket ID a um endereço para que os processos possam comunicar.

sockaddr

- Sockets específicos para a internet:

```
struct sockaddr_in {
    sa_family_t     sin_family; //AF_INET
    in_port_t       sin_port; //número da porta
    struct in_addr  sin_addr; //endereço IP
}
```

- Sockets em UNIX (só funciona entre processos na mesma máquina):

```
struct sockaddr_un {
    uint8_t         sun_lenght;
    short           sun_family; //AF_LOCAL
    char            sun_path[100];
}
```

listen

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


int listen (int sid, int size);
```

- Retorna 0 em caso de sucesso, ou -1 em caso de falha
- size é o número permitido de conexões pendentes (geralmente em Unix é limitado a 5)

accept

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


int accept (int sid, struct sockaddr *addrPtr, int *lenPtr);
```

- Retorna o ID do socket e o endereço do cliente conectado ao socket.
- lenPtr é o endereço do cliente ativo e addrPtr é o tamanho desse endereço
- Espera por um pedido de conexão e quando recebe uma, cria um socket para essa comunicação

send

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

int send (int sid, const char *bufferPtr,
          int len, int flag);
```

- Envia uma mensagem
- Retorna o número de bytes enviados ou -1 em caso de falha
- *bufferPtr é o buffer recebido, len é o tamanho do buffer em bytes e flag são opções especiais (geralmente 0)

recv

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

int recv (int sid, const char *bufferPtr,
          int len, int flag);
```

- Recebe uma mensagem através do buffer bufferPtr com len bytes
- Retorna o número de bytes recebidos em caso de sucesso, ou -1 em caso de falha

shutdown

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


int shutdown (int sid, int how);
```

- Funciona como um fecho parcial
- Desactiva o envio (how=1 ou how=2) ou a receção (how=0 ou how=2)
- Retorna -1 em caso de falha

connect

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


int connect (int sid, struct sockaddr *addrPtr, int len);
```

- É a primeira função que o cliente chama
- Específica o destino da conexão (addrPtr) e retorna 0 em caso de sucesso e -1 em caso de falha

Conexões

- Uma conexão é definida por:
  - IP de
  - Porta de
  - Protocolo
  - IP para
  - Porta para
- Assim várias conexões para partilhar o mesmo IP e porta

Portas

- O iniciador da comunicação necessita de uma porta fixa para poder comunicar
- Isto significa que algumas portas devem estar reservadas para certos serviços (por exemplo, portas 20 e 21 para o FTP, 80 para HTTP, etc.)

API para gerir nomes e endereços IP

- Estrutura hostent - descreve o IP e os nomes do host
- gethostbyname - host de uma máquina específica
- htons, htonl, ntohs, ntohl - ordem dos bytes
- inet_pton, inet_ntop - conversão dos número do endereço IP do formato de apresentação para o formato de rede e vice-versa

gethostname

```
#include <unistd.h>

int gethostname (char *hostname, size_t nameLenght);
```

- Retorna o nome da máquina (hostname) onde este comando é executado
- Retorna -1 em caso de falha

Estrutura hostent

```
struct hostent {
    char *h_name; //nome oficial do host
    char **h_aliases; //hostnames alternativos
    char h_addrtype; //tipo de endereço do host AF_INET ou AF_INET6
    char h_lenght; //4 ou 16 bytes
    char **h_addr_list; //lista de endereços IPv4 ou IPv6
}
```

inet_pton

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

int inet_pton (int family, const char *strPtr,
               void *addrPtr);
```

- Retorna 1 em caso de sucesso, 0 em caso de erro de apresentação e -1 em caso de erro
- family pode ser AF_INET ou AF_INET6
- strPtr é o endereço IP em string com pontos (por exemplo, 193.168.1.1)
- addrPtr aponta para o resultado 32 bit (AF_INET) ou 128 bit (AF_INET6)

inet_ntop

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

int inet_ntop (int family, const char *addrPtr,
               char *strPtr, size_t len);
```

- Retorna 1 em caso de sucesso, 0 em caso de erro de apresentação e -1 em caso de erro
- family pode ser AF_INET ou AF_INET6
- strPtr é o endereço IP em string com pontos (por exemplo, 193.168.1.1)
- addrPtr aponta para o resultado 32 bit (AF_INET) ou 128 bit (AF_INET6)
- len é o tamanho do destino

Exemplo: Código de um Servidor TCP/IP

```
sockaddr_in serverAddr;
sockaddr &serverAddrCast = (sockaddr &) serverAddr;

// tcp/ip socket
int listenFd = socket(AF_INET, SOCK_STREAM, 0);

bzero(&serverAddr, sizeof(serverAddr));
serverAddr.sin_family = AF_INET;
// qualquer interface internet neste servidor
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
serverAddr.sin_port = htons(13);

bind(listenFd, &serverAddrCast, sizeof(serverAddr));

listen(listenFd, 5);

for ( ; ; ) {
    int connectFd = accept(listenFd, (sockaddr *) NULL, NULL);
    //... operações read e write no connectFd
    shutdown(connectFd, 2);
    close(connectFd);
}
```

- Servidor iterativo, apenas recebe uma conexão de cada vez

Exemplo: Código de um Cliente TCP/IP

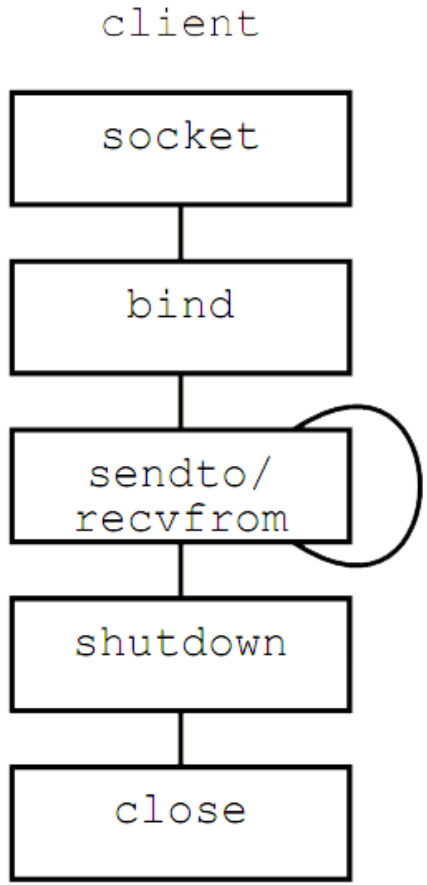
```
sockaddr_in serverAddr;
sockaddr &serverAddrCast = (sockaddr &) serverAddr;

// tcp/ip socket
int socketFd = socket(AF_INET, SOCK_STREAM, 0);

bzero(&serverAddr, sizeof(serverAddr));
serverAddr.sin_family = AF_INET;
// host IP em formato decimal com pontos
inet_pton(AF_INET, serverName, serverAddr.sin_addr);
serverAddr.sin_port = htons(13);

connect(sockFd, serverAddrCast, sizeof(serverAddr));
    //... operações read e write no sockFd
shutdown(sockFd, 2);
close(sockFd);
```

Comunicação sem conexão

A comunicação é simétrica (peer-to-peer)

- socket
- bind - é opcional para o iniciador
- sendto, recvfrom - (repetido)
- shutdown
- close

Comunicação sem conexão



Variações UDP

- Não é necessário que os dois sockets façam bind
  - O recetor recebe o endereço do emissor
- É possível que um socket UDP faça connect
  - Neste caso, deve ser usado send/recv em vez de sendto/recvfrom

sendto

Para protocolos sem conexão

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

int sendto (int sid, const void *bufferPtr,
            size_t bufferLength, int flag,
            struct sockaddr *addrPtr, socklen_t addrLength);
```

- Envia uma mensagem (bufferPtr) de tamanho bufferLength para o endereço especificado por addrPtr de tamanho addrLength
- Retorna o número de bytes enviados em caso de sucesso ou -1 em caso de erro

recvfrom

Para protocolos sem conexão

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

int recvfrom (int sid, void *bufferPtr, int bufferLength,
              int flag, sockaddr *addrPtr, int *addrLengthPtr);
```

- Recebe uma mensagem (bufferPtr) de tamanho máximo bufferLength do endereço especificado por addrPtr de tamanho *addrLengthPtr
- Retorna o número de bytes recebidos em caso de sucesso ou -1 em caso de erro

Exemplo: Código de um Servidor UDP

```
int socketId = socket(AF_INET, SOCK_DGRAM, 0);

sockaddr_in serverAddr, clientAddr;
sockaddr &serverAddrCast = (sockaddr &) serverAddr;
sockaddr &clientAddrCast = (sockaddr &) clientAddr;

//permite conexões para qualquer endereço do host
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(serverPort);
serverAddr.sin_addr.s_addr = INADDR_ANY;

//associa o processo com a porta
bind(socketId, &serverAddrCast, sizeof(addr));

//recebe de um cliente
int size = sizeof(clientAddr);
recvfrom(socketId, buffer, bufferSize, 0, clientAddrCast, &size);

//envia para o cliente (o que acabou de enviar mensagem)
sendto(socketId, buffer, bufferSize, 0, clientAddrCast, size);

close(socketId);
```

Exemplo: Código de um Cliente UDP

```
int socketId = socket(AF_INET, SOCK_DGRAM, 0);

sockaddr_in serverAddr, clientAddr;
sockaddr &serverAddrCast = (sockaddr &) serverAddr;
sockaddr &clientAddrCast = (sockaddr &) clientAddr;

//especifica o endereço e porta do servidor
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(serverPort);
struct hostent *hp = gethostbyname(hostName);
memcpy((char*)&serverAddr.sin_addr, (char*)hp->h_addr, hp->h_length);

// não é necessário fazer bind se não for peer-to-peer
int size = sizeof(clientAddr);
sendto(socketId, buffer, bufferSize, 0, clientAddrCast, &size);

recvfrom(socketId, buffer, bufferSize, 0, clientAddrCast, size);

close(socketId);
```