Curso de C/C++ Avançado

Aula 9 – Sockets

Allan Lima – http://allanlima.wordpress.com

- Você pode:
  - copiar, distribuir, exibir e executar a obra
  - criar obras derivadas
  - fazer uso comercial da obra
- Sob as seguintes condições:
  - **Atribuição**. Você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante.
  - **Compartilhamento pela mesma Licença**. Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.
  - Para cada novo uso ou distribuição, você deve deixar claro para outros os termos da licença desta obra.
  - Qualquer uma destas condições podem ser renunciadas, desde que Você obtenha permissão do autor.
- **Veja aqui a licença completa**

Sockets

- É um mecanismo que os programas utilizam para se comunicar
- Amplamente utilizados em aplicações dos mais diversos tipos

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*)

TCP X UDP

TCP	UDP
Orientado à conexão	Não possui conexão
Confiável	Não Confiável
Entrega ordenada	Entrega sem garantias de ordem
Controle de Fluxo	Sem controle de fluxo

Winsock

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

Inicialização

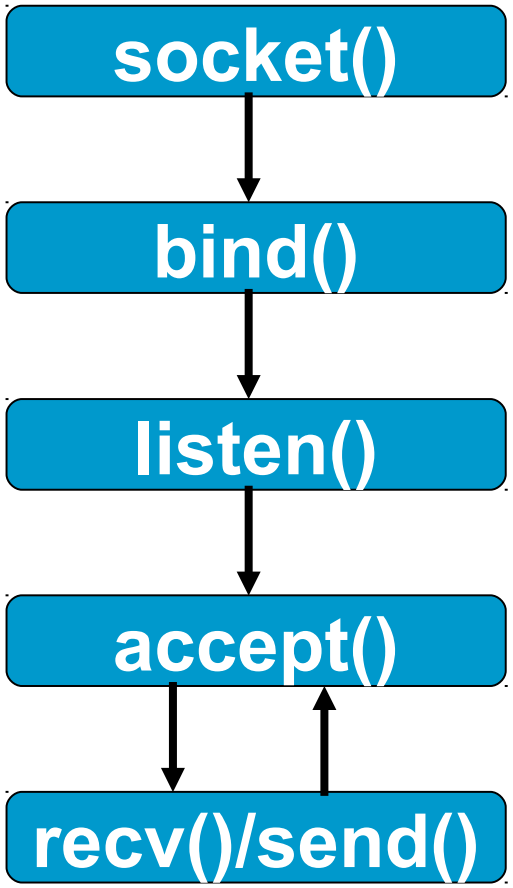
- Antes de usarmos as funções da Winsock devemos inicializa-la chamando a função WSAStartup

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

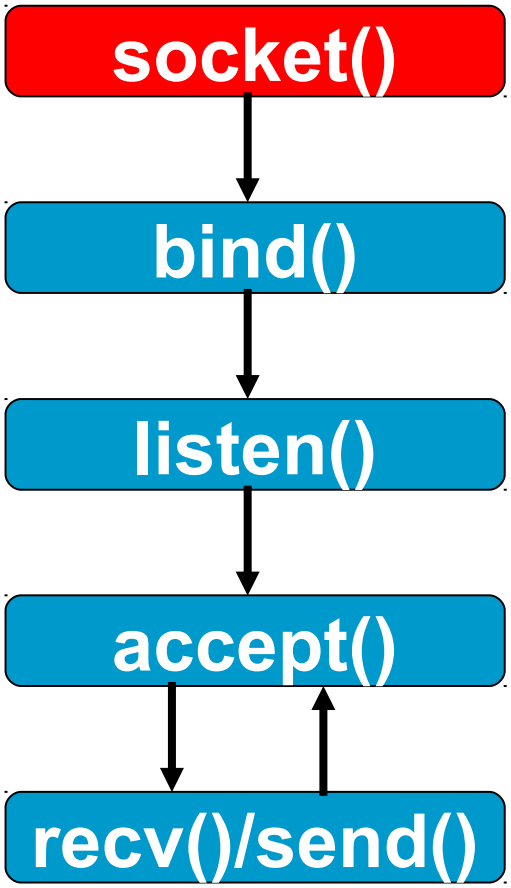
Associa o socket a uma porta

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

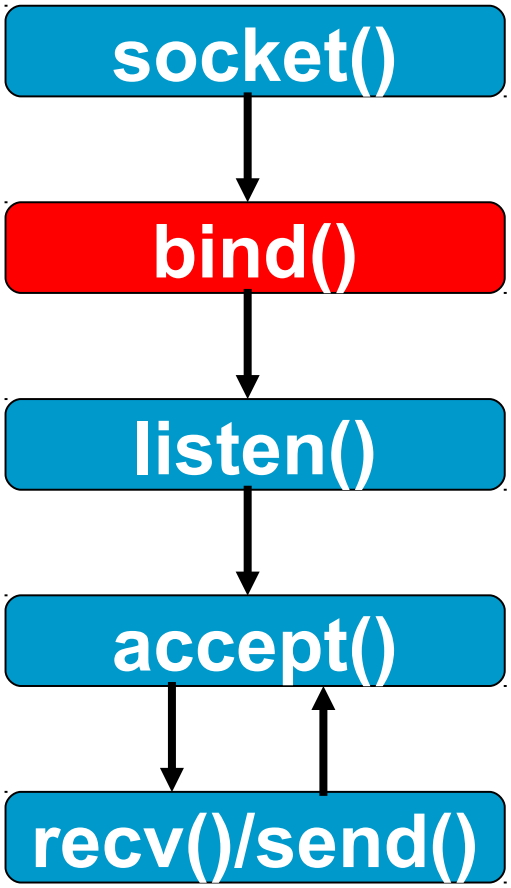
Envia e recebe dados

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

Um pouco de História

- Para atrair os programadores do Unix a API de sockets do windows é muito parecida com ela
- Herdando até alguns dos problemas
- Por exemplo a struct sockaddr

struct sockaddr

```
struct sockaddr {
    // família do endereço, AF_xxx
    u_short sa_family;
    // 14 bytes do endereço do protocolo
    char sa_data[14];
};
```

- Pode guardar informação sobre diversos tipos de sockets
- Não é muito utilizada, mas muitas funções a recebem como parâmetro
- Normalmente fazemos um cast para este tipo quando chamamos alguma função biblioteca

sockaddr_in

```
struct sockaddr_in { // in de internet
    u_short sin_family; // família do endereço
    u_short sin_port; // porta
    struct in_addr sin_addr; // endereço IP
    char sin_zero[8]; // completa o tamanho
};
```

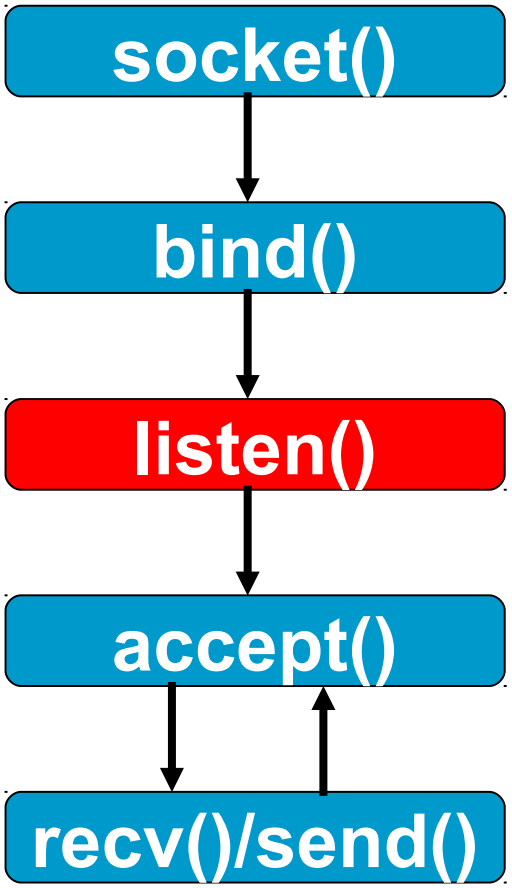
- Agrupa os mesmos dados da sockaddr
- Criada para resolver as dificuldades de manipulação do endereço
- sin_zero deve ser setado com zeros através da função memset()

struct in_addr

```
typedef struct in_addr {
    union {
        struct {
            u_char s_b1,s_b2,s_b3,s_b4;
        } S_un_b;
        struct {
            u_short s_w1,s_w2;
        } S_un_w;
        u_long S_addr;
    } S_un;
} in_addr;
```

- Une as múltiplas formas de se representar um endereço IP

Configurando a fila



Cria o socket

Associa o socket a uma porta

Configura a lista de conexões

Recebe uma conexão de um cliente

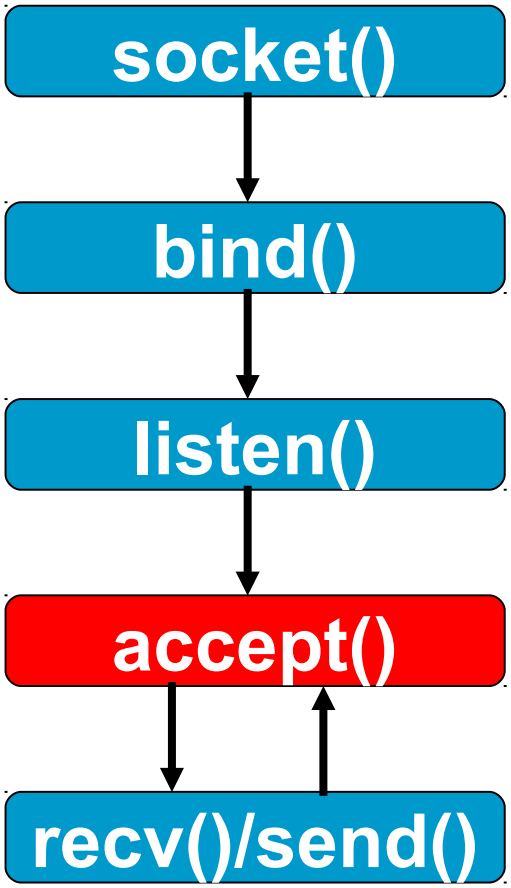
Envia e recebe dados

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

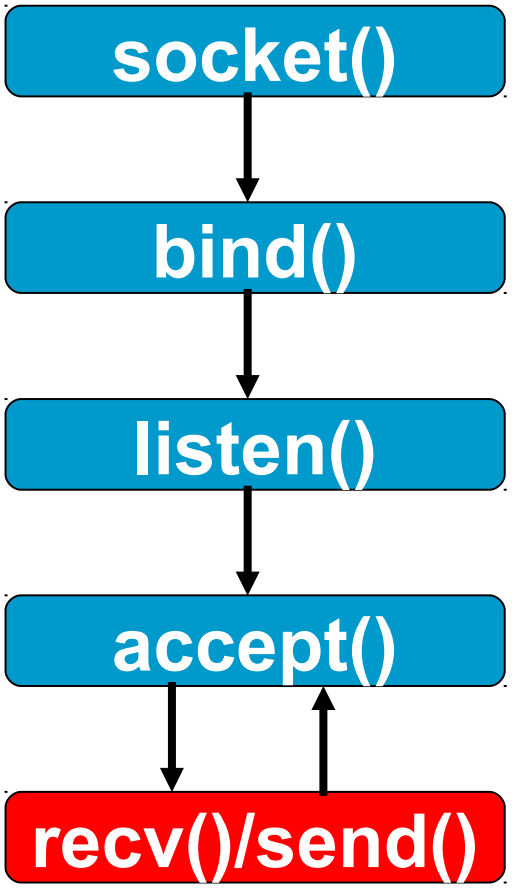
Envia e recebe dados

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

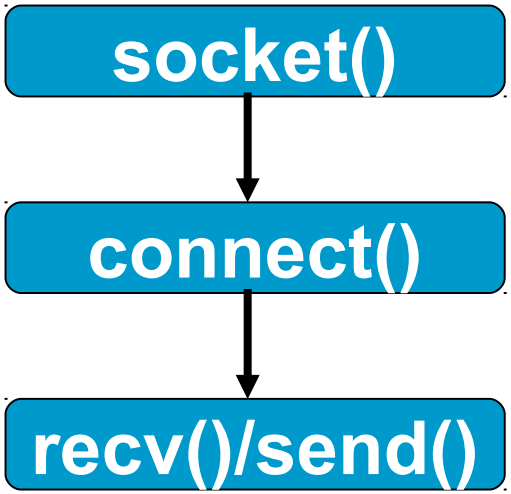
```
int send(
   SOCKET s, const char* buf,
   int len, int flags
);
```

- *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

- exemploServidorTCP.cpp

Criando um cliente TCP

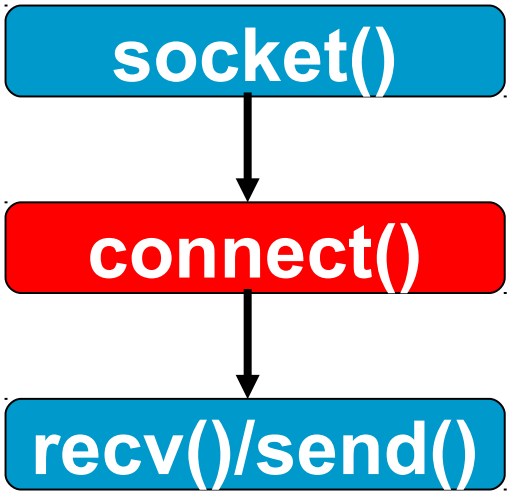


Cria o socket

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

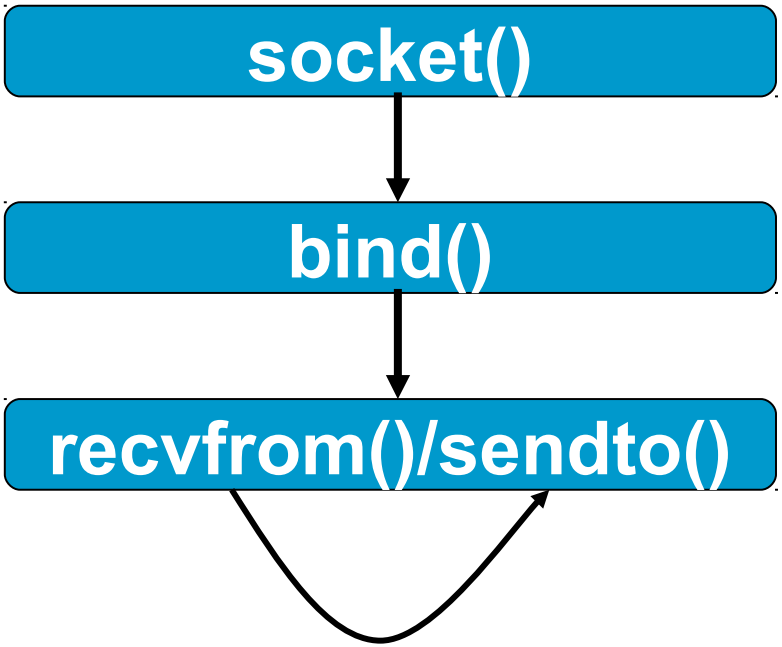
```
int connect(
    SOCKET s,
    const struct sockaddr* name,
    int namelen
);
```

- *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

- exemploClienteTCP.cpp

Criando um servidor UDP

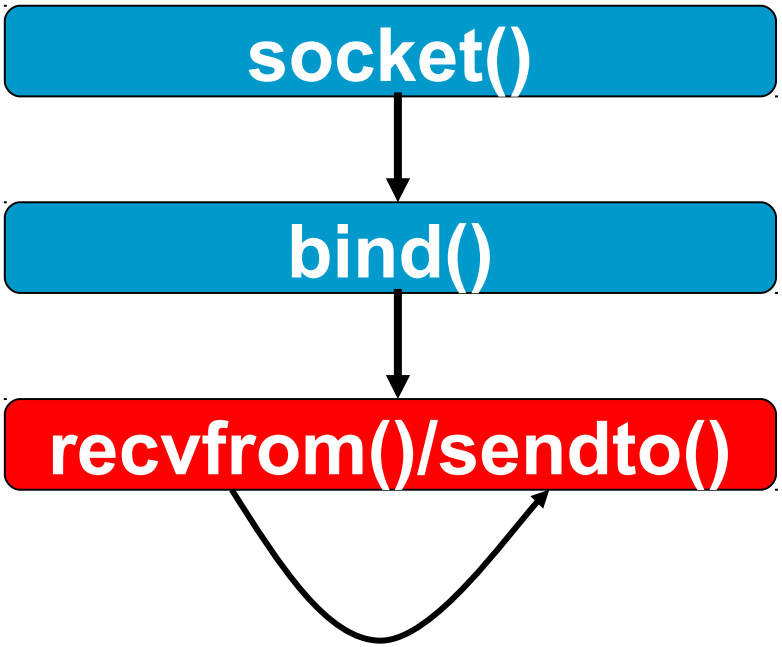


Cria o socket

Associa o socket a uma porta

Envia e recebe dados

Enviando e recebendo dados



Cria o socket

Associa o socket a uma porta

Envia e recebe dados

Recebendo dados

```
int recvfrom(
        SOCKET s,
        char* buf,
        int len,
        int flags,
        struct sockaddr* from,
        int* fromlen
);
```

Recebendo dados

- *s* é o identificador do socket
- *buf* é o local onde os dados recebidos serão guardardados
- *len* é o número máximo de bytes que serão lidos
- *flags* influenciam o comportamento da função
  - *MSG_PEEK*, MSG_OOB
- *from* irá guardar os dados do cliente
- *fromlen* é o tamanho de from
- Retorna o número de bytes recebidos em caso de sucesso ou SOCKET_ERROR em caso de falha.

Enviando dados

```
int sendto(
    SOCKET s,
    const char* buf,
    int len,
    int flags,
    const struct sockaddr* to,
    int tolen
);
```

Exemplo

- exemploServidorUDP.c

Criando um cliente UDP

- É praticamente idêntico ao servidor
- Exemplo
  - exemploClienteUDP.cpp

Enviando dados

- *s* é o identificador do socket
- *buf* contém os dados a serem enviados
- *len* é o número máximo de bytes que serão enviados
- *flags* influenciam o comportamento da função
  - *MSG_DONTROUTE,* MSG_OOB
- *to* guarda o endereço do cliente
- *tolen é o tamanho de to*
- Retorna o número de bytes enviado em caso de sucesso ou SOCKET_ERROR em caso de falha.

Sockets e a MFC

- A MFC possui um grande conjunto de classes que dão suporte a sockets
- Principais Classes:
  - CSocket
  - CSocketFile
  - CArchive
- Estas classes possuem um alto nível de abstração e encapsulam a API s do Windows

Exemplos

- exemploServidorTCPSimples.cpp
- exemploClienteTCPSimples.cpp
- exemploServidorTCPArquivo.cpp
- exemploClienteTCPArquivo.cpp
- exemploServidorUDP.cpp
- exemploClienteUDP.cpp

Exercícios

1) Implemente um servidor que envia arquivos para os seus clientes. O cliente envia o nome do arquivo para o servidor e este envia os bytes no arquivo para o cliente.

Referências

- Class Library Reference
  - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_mfc_class_library_reference_introduction.asp
- Winsocks utilizando Visual C++
  - http://200.250.4.4/curso-redes-graduacao/tutorial/socket/tutwsk_c.html
- Johnnie's Winsock Tutorial
  - http://windows.ittoolbox.com/browse.asp?c=WindowsPeerPublishing&r=http%3A%2F%2Fwww%2Ehal%2Dpc%2Eorg%2F%7Ejohnnie2%2Fwinsock%2Ehtml