



MC833AB – Programação de Redes de Computadores

Professor Nelson Fonseca

<http://www.lrc.ic.unicamp.br/mc833/>

Roteiro

- **Objetivo: compreender os códigos de um servidor e de um cliente TCP usando sockets (Capítulos 2, 3 e 4 do livro texto)**
- Sockets em SOs Unix-like
- Algoritmo do cliente e do servidor TCP
- Funções importantes para clientes e servidores TCP (mais uma vez, em C)
- Estrutura de endereçamento dos sockets (tudo em C)
- Atividade prática

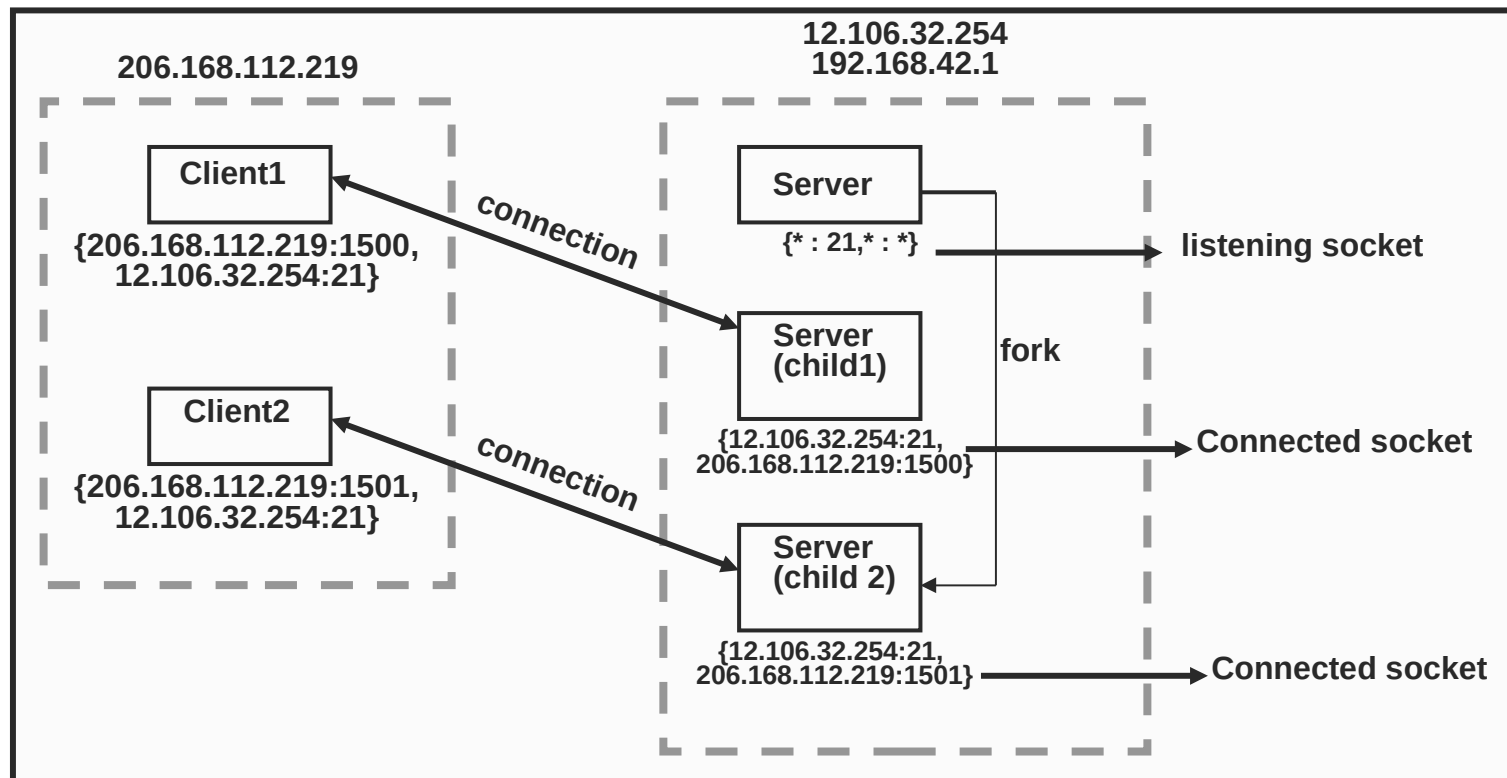
[Sockets]

API Sockets

- Em SOs Unix-like, tudo é arquivo!
- Socket = “o arquivo para comunicação de programas via rede”
- cliente/servidor
- Dois tipos de serviço de transporte via API Sockets
 - Datagrama, entrega não confiável
 - Fluxo de bytes, entrega confiável

Pares de sockets

- 4-tupla <endereço IP local, porta local, endereço IP remoto, porta remota>
- Exemplo de conexão de dois clientes a um servidor:



[Pares de sockets (Portas em SOs Unix-like)]

- No lado do servidor, precisa definir portas > 1023 (não root)
- No lado do cliente, é automático: Portas temporárias
- Dúvida sobre portas e serviços?
 - `/etc/services`

[Pares de sockets (Endereços em SOs Unix-like)]

- `/sbin/ifconfig`
- No lado do servidor a escolha pode ser automática
 - Mais necessário caso hajam múltiplos endereços e se quiser restringir o funcionamento do servidor

[Então, sockets possibilitam...]

- Uma comunicação entre **dois processos** via TCP (ou UDP) identificada univocamente por dois pares de valores:
 - um “socket local” = (IP local, porta local), e
 - um “socket remoto” = (IP remoto, porta remota)
- Comunicação Cliente - Servidor:
 - máquina Cliente denominada “local”,
 - máquina Servidora denominada “remota”
- **Abstração dos detalhes das camadas inferiores!!!**

[Como seria um servidor TCP?]

- Cria `s = socket (porta); //(porta > 1023)`
- Informa que `s` é um servidor; // Deve esperar conexões;
- enquanto (1)
 - Aguarda conexão dos clientes;
 - Transfere dados para o cliente;
 - Fecha a conexão;
- `sai();`
- **Obs.: Sem concorrência**

[Como seria um cliente TCP?]

- Cria `s = socket ()`;
- Conecta `s` ao servidor; // Precisa saber IP e porta
- Transfere dados do servidor;
- `sai()`;

[Função socket]

```
int socket ( int family, int type, int protocol )  
retorna > 0 se OK, -1 se erro
```

- Retorna um descritor de socket (um inteiro positivo pequeno)
- Muito semelhante a um descritor de arquivo Unix (Suporta, por exemplo, operações de read, write e close)
- **cliente** e **servidor** usam

Função socket (2)

■ Parâmetros:

family: uma dentre as constantes:

AF_INET - socket usa internet IPv4

AF_INET6 - socket usa internet IPv6

AF_UNIX ou AF_LOCAL - socket domínio Unix

type: um dentre as constantes:

SOCK_STREAM: socket será usado com TCP

SOCK_DGRAM: socket será usado com UDP

protocol: 0 para aplicações comuns

Função socket (3)

- Estaremos interessados apenas nas combinações:

<i>Família</i>	<i>Tipo</i>	
AF_INET	SOCK_STREAM	SOCK_DGRAM
AF_UNIX	SOCK_STREAM	SOCK_DGRAM

Exemplo:

```
sockfd = socket(AF_INET, SOCK_STREAM, 0),
```

cria um socket Internet (IP4) para uso com TCP.

Função connect

```
int connect ( int sockfd, const struct sockaddr *servaddr,  
              int addrlen)  
            retorna 0 se OK, -1 se erro,
```

- **cliente** usa para iniciar conexão com servidor remoto:
- "3 way handshake"
- sockfd: obtido na chamada anterior a socket ()
- servaddr: estrutura inicializada previamente com a identificação do socket remoto: (IP remoto, # porta remota)
- Não é necessário que o cliente chame bind :
o socket local escolhido pelo kernel e consiste do par:
(IP local, # porta transiente), escolhida de forma a não conflitar com outras em uso

[Função bind]

```
int bind(int sockfd, (struct sockaddr)* myaddr, int socklen)
```

retorna 0 se OK, -1 se erro

`bind` associa ao descritor `sockfd` um valor para o “socket local” passado na estrutura `myaddr`.

A aplicação tem a opção de deixar para o sistema determinar o # IP ou o # porta ou ambos;

Servidor usa para associar o socket à um IP/porta (o cliente não precisa pois isso é feito pelo SO somente na conexão)

Função bind (2)

- Isto em geral é conveniente, pois se o host tiver mais de um endereço, o “mais apropriado” para a comunicação é escolhido (no netstat “0.0.0.0”)
- Se o sistema escolhe a porta, esta não conflitará com nenhuma outra (temporária).
- Para deixar o sistema escolher o endereço deve-se atribuir `INADDR_ANY` no campo de endereço da estrutura *myaddr*
- Para deixar o sistema escolher a porta deve-se atribuir 0 ao campo de porta da estrutura *myaddr*:

```
*myaddr.sin_port = 0;
```

```
*myaddr.sin_addr.s_addr = INADDR_ANY
```

[Função `listen`]

- Um socket é considerado *ativo* se a aplicação invoca *connect* com o socket, iniciando o “3-way handshake” com outro host;
- se a aplicação invocar *listen*, então o socket passa a ser *passivo* (aceita conexões);
- **Servidores** devem usar para modificar o socket (clientes invocam *connect* enquanto servidores invocam *listen* seguido de *accept*).

```
int listen (int sockfd, (struct sockaddr) * myaddr, int backlog)
```

retorna 0 se OK, -1 se erro

O parâmetro *backlog* corresponde ao tamanho de uma fila no kernel para o número de conexões em andamento e completadas.

Função accept

- Invocada por um **servidor** TCP para obter os dados e retirar da fila a 1ª conexão da fila de conexões concluídas.

```
int accept (int sockfd, (struct sockaddr) * cliaddr, int * socklen)
```

retorna: valor descritor de socket (>0) se OK, -1 se erro

socklen é passado por valor-resultado, retornando o tamanho real da estrutura preenchida pelo kernel (igual ao passado, no caso de um socket internet)

[Estrutura sockaddr]

```
struct sockaddr {  
    unsigned short sa_family; // 2 bytes - família (AF_xxx)  
    char           sa_data[14]; // 14 bytes - endereço  
};
```

- Estrutura genérica para manter dados de endereços
- Mesmas famílias do socket
- Para facilitar a manipulação do `sa_data`, usa-se `sockaddr_in`

[Estrutura sockaddr_in]

```
struct sockaddr_in {  
    short int     sin_family;   // 2 bytes  
    unsigned short int sin_port; // 2 bytes  
    struct in_addr sin_addr;    // 4 bytes  
    unsigned char  sin_zero[8]; // igualar o tamanho  
};
```

- Estrutura para Ipv4 (**internet**)
- A porta e o endereço devem ser manipuladas por funções especiais (ex: `inet_pton`, `htonl`, `htons`, `inet_ntop`)
- Pode fazer cast para `sockaddr`

Atividade prática

- Familiarizar-se com as funções de sockets e com as ferramentas do SO através da análise dos códigos de um servidor e de um cliente TCP
- Site da atividade:
http://www.lrc.ic.unicamp.br/mc833/exercicios/socket_introducao/index.html

Próxima aula

- Implementação de um cliente TCP e de um servidor que aceite conexões concorrentes