



# MC833AB - Programação de Redes de Computadores

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

# Roteiro

- **Objetivo: Explicar o funcionamento de um servidor concorrente com sockets (Capítulos 3, 4 e 5 do livro texto)**
- Conversão dos formatos de endereço e de porta (reforçando)
- Servidor TCP concorrente
- Funções importantes para suporte à concorrência
- Atividade prática

# Conversões de formato de porta e endereço

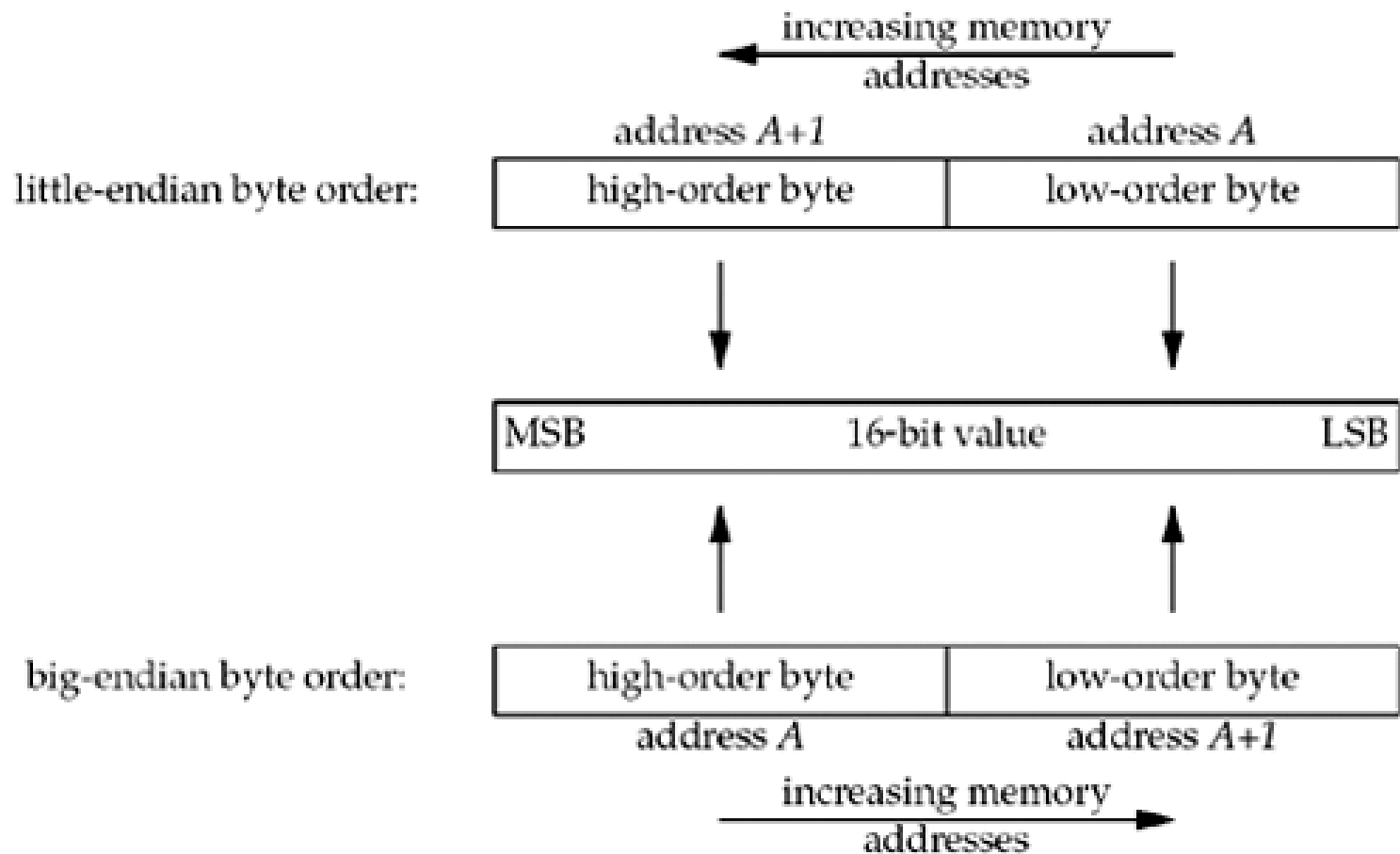
```
struct sockaddr_in {  
    short int     sin_family;   // 2 bytes  
    unsigned short int sin_port; // 2 bytes  
    struct in_addr sin_addr;    // 4 bytes  
    unsigned char  sin_zero[8]; // igualar o tamanho  
};
```

- **A porta e o endereço devem ser manipulados por funções especiais (ex: `inet_pton`, `htonl`, `htons`, `inet_ntop`)**
- `inet_*` : funções para ler/escrever endereços IP de/para a estrutura
- `htonl`, `htons`, `ntohl`, `ntohs`

# [ Sem converter antes... ]

- Ordem dos bytes armazenados internamente nos hosts pode ser diferente da ordem dos bytes definidos pelo TCP/IP
- little endian X big endian
- network byte order = big endian
- host byte order = ???
  - GNU/Linux em um i386 = little endian

# [ Little endian X Big endian ]



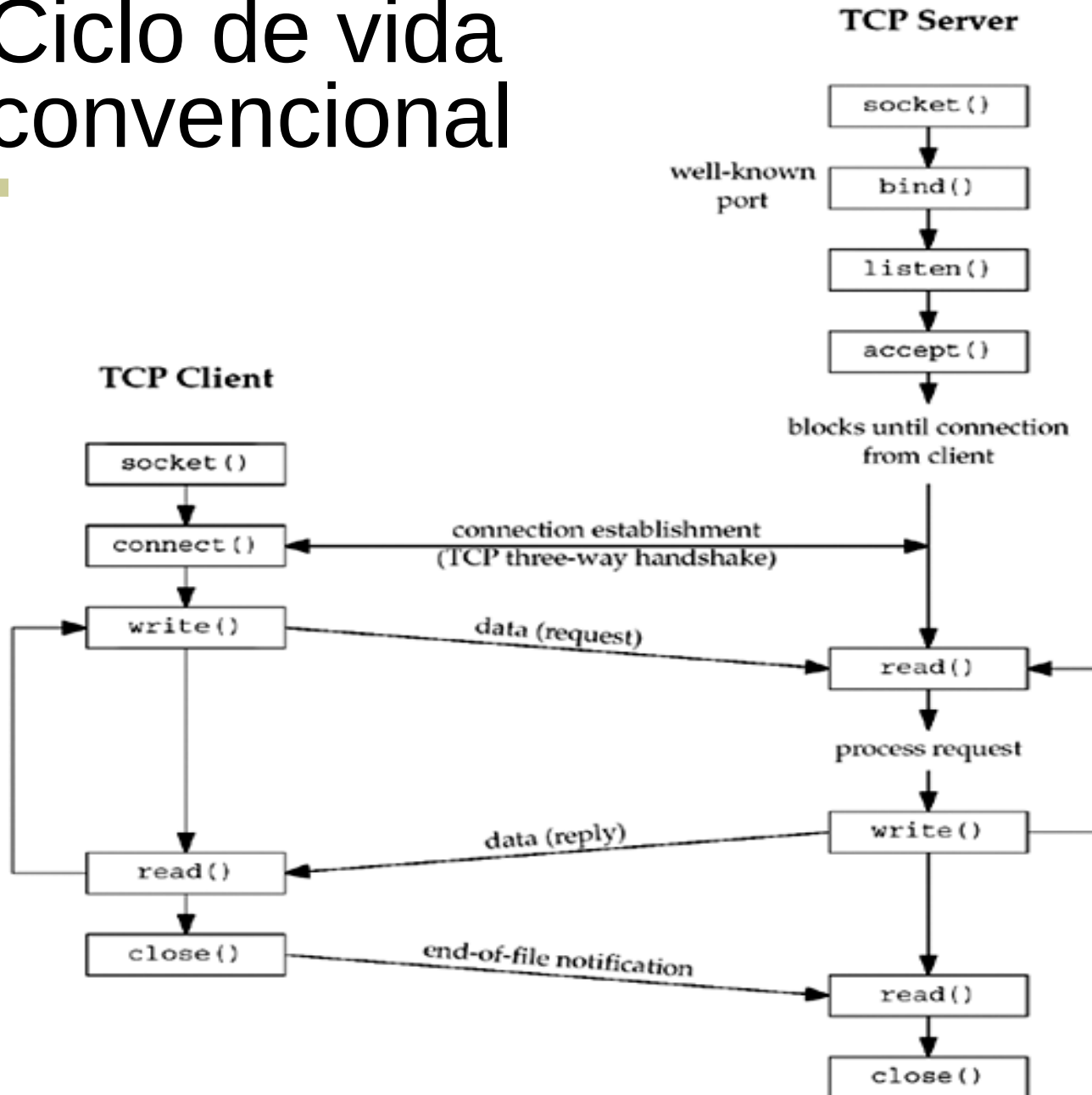
# [ Sem converter antes... ]

- 1027 (decimal)
- 00000100000000011 (binário)
- Little endian:
  - Endereço 00: 00000011
  - Endereço 01: 00000100
- Big endian:
  - Endereço 00: 00000100
  - Endereço 01: 00000011
- Porta 1027 armazenada em um socket seria lida como 772

# [ Um servidor, várias conexões ]

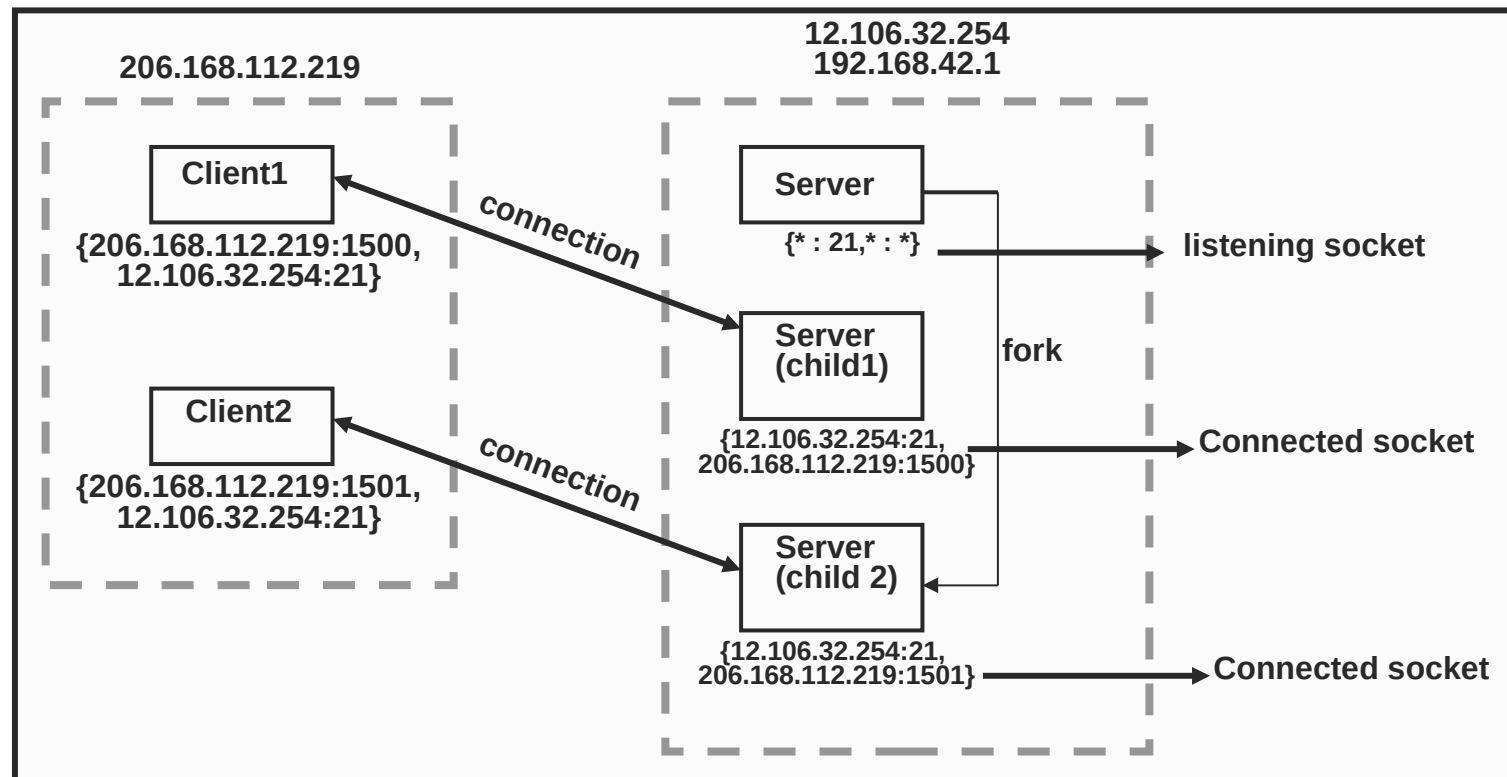
- Em geral o servidor cria um novo processo para tratar a nova conexão, passando para o mesmo o descritor do “socket conectado”.
- Nesse caso o servidor pode tratar simultaneamente várias conexões (com durações variáveis ou imprevisíveis) e dizemos que ele é um “servidor concorrente”.

# Ciclo de vida convencional





# Servidor TCP concorrente



# Mudanças no algoritmo do servidor

- Cria `s = socket (porta); //(porta > 1023)`
- Informa que `s` é um servidor; // Deve esperar conexões;
- enquanto (1)
  - Aguarda conexão dos clientes;
  - Cria um novo processo;**
  - se for o processo criado // Processo FILHO**
    - Transfere/Recebe dados para/do cliente;
    - Fecha a conexão;
  - senão // Processo PAI**
    - Volta a aguardar novos clientes;**
- `sai();`

# [fork]

```
# include <unistd.h>  
pid_t fork(void);
```

Retorna: 0 no filho, ID do processo no pai, -1 se erro

- O fork retorna o identificador do processo filho ao pai (process ID) e o valor 0 (process ID) ao filho
- Os descritores abertos antes do fork são compartilhados com os filhos
- o “connected socket” após um accept seguido de fork é compartilhado com o filho
- Dois usos do fork: executa cópia idêntica (fork) ou executa outro programa (fork e exec)

# [exec

```
#include <unistd.h>
```

```
int execl (const char *pathname, const char *arg0, ... /* (char *) 0 */ );
```

```
int execv (const char *pathname, char *const argv[ ]);
```

```
int execl (const char *pathname, const char *arg0, ...  
          /* (char *) 0, char *const envp[ ] */ );
```

```
int execve (const char *pathname, char *const argv[ ], char *const envp[ ]);
```

```
int execlp (const char *filename, const char *arg0, ... /* (char *) 0 */ );
```

```
int execvp (const char *filename, char *const argv[ ]);
```

All six return: -1 on error, no return on success

# [ Detalhes da implementação ]

- após **accept** e **fork**, o processo filho deve manipular o socket conectado e o pai continuar a escutar no “listening socket”
- **close** decrementa contador de referências
- FYN é enviado **somente** quando contador de referências possui valor zero

# [ Detalhes da implementação ]

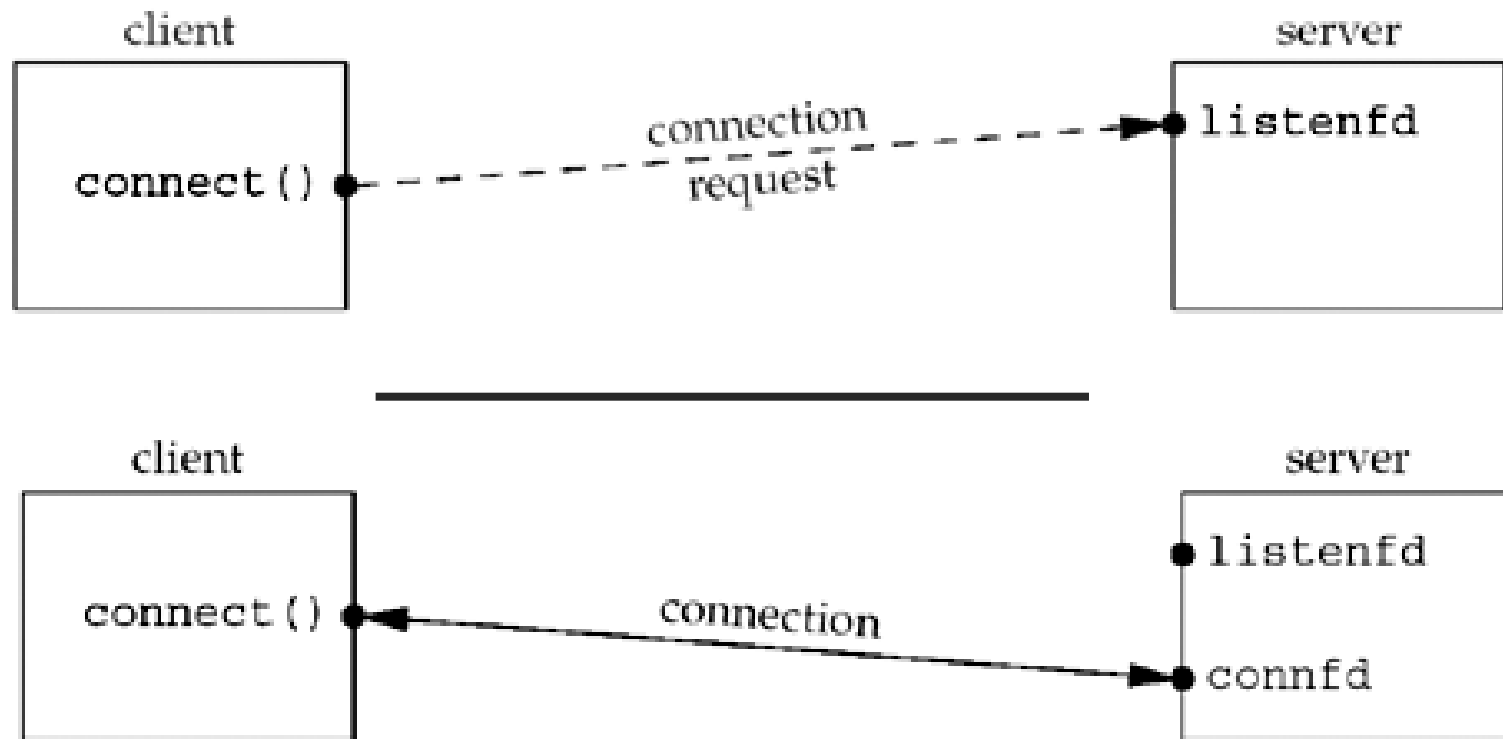
```
pid_t pid;  
int listenfd, connfd;
```

```
listenfd = Socket( ... ); /* fill in sockaddr_in{ } with server's well-known port */
```

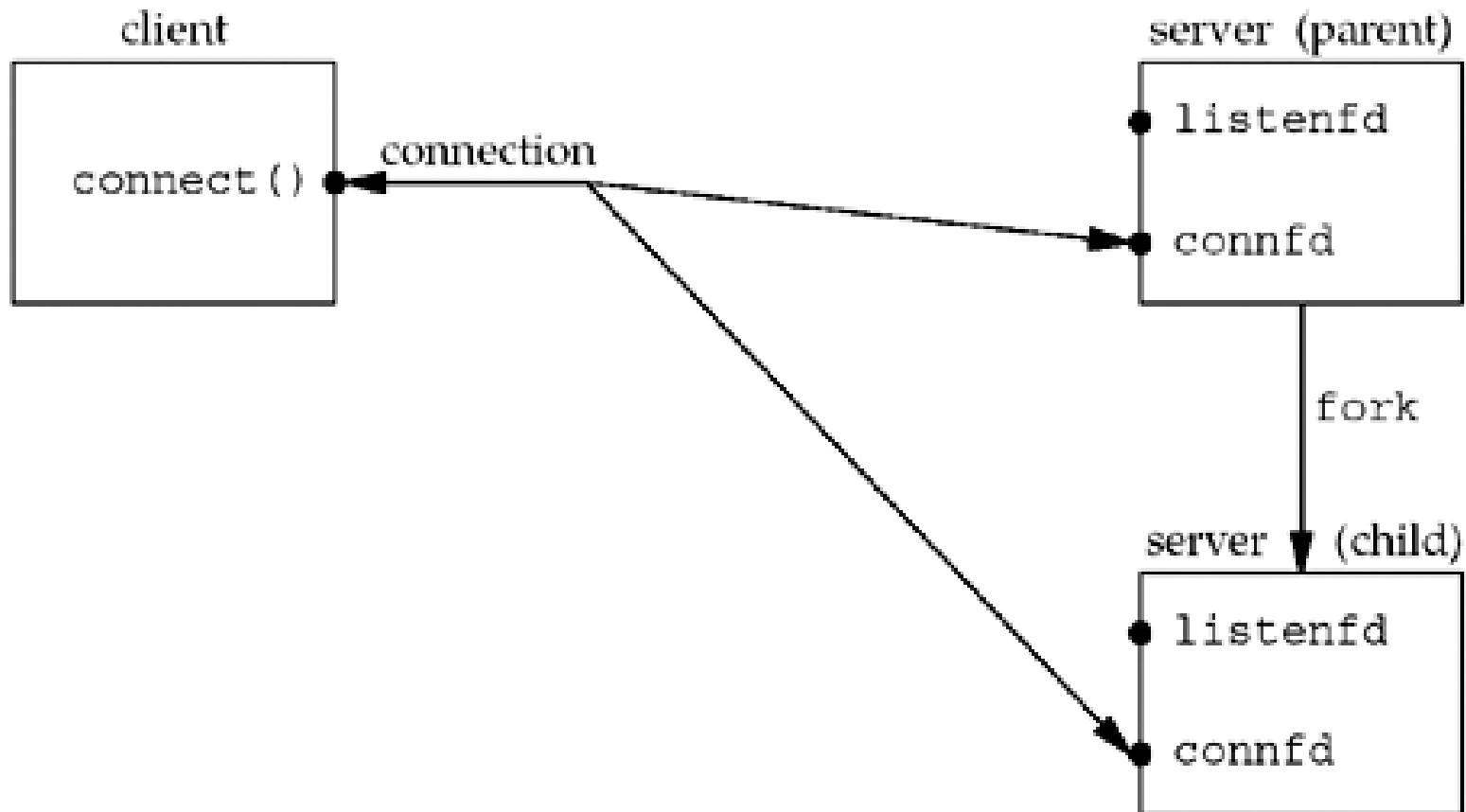
```
Bind(listenfd, ... );  
Listen(listenfd, LISTENQ);
```

```
for ( ; ; ) {  
    connfd = Accept (listenfd, ... ); /* probably blocks */  
  
    if( (pid = Fork()) == 0 ) {  
        Close(listenfd); /* child closes listening socket */  
        doit(connfd); /* process the request */  
        Close(connfd); /* done with this client */  
        exit(0); /* child terminates */  
    }  
  
    Close(connfd); /* parent closes connected socket */  
}
```

# [Funcionamento]

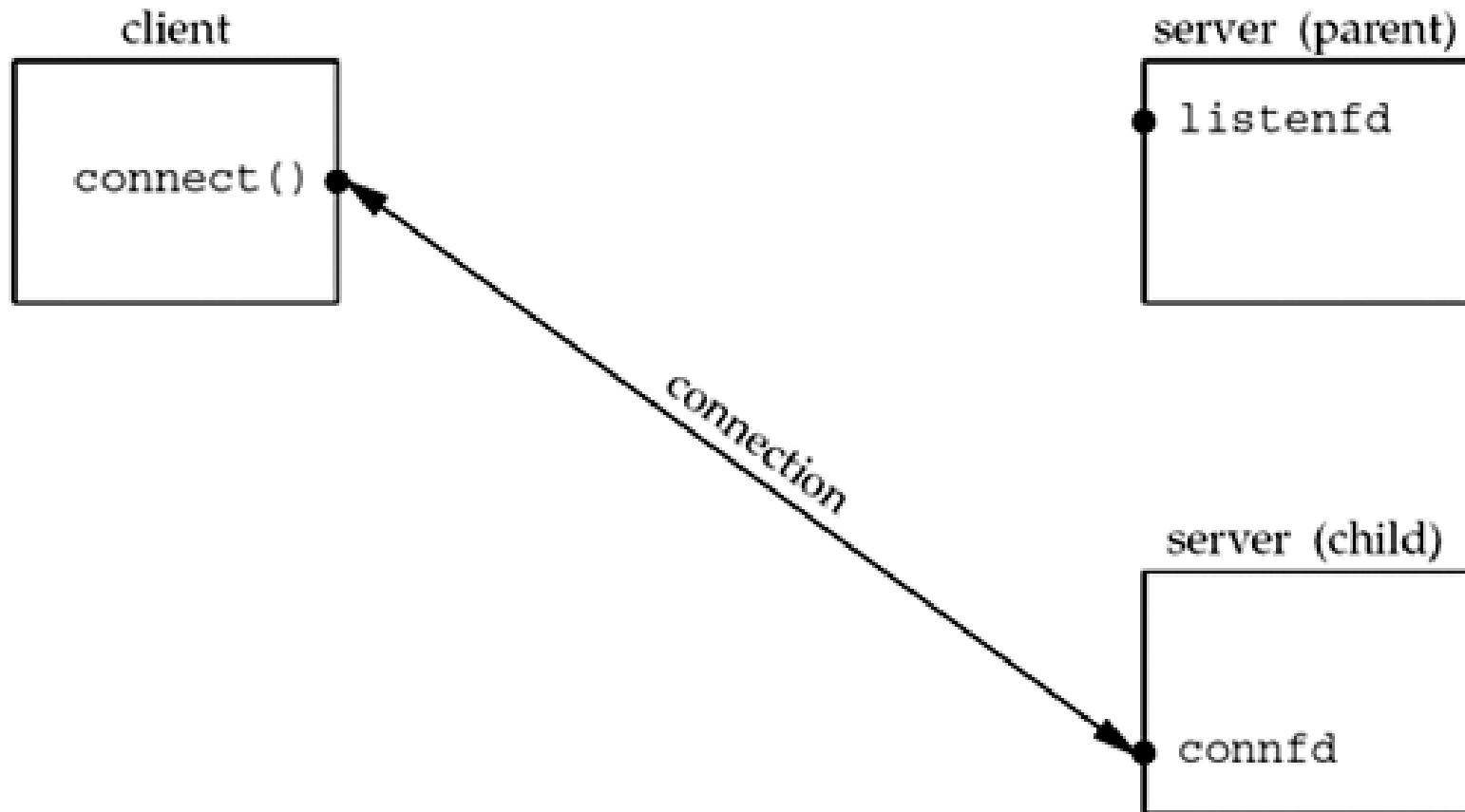


# [Funcionamento]





# [Funcionamento]



# Atividade prática

- Escrever um servidor TCP concorrente (“eco” o conteúdo enviado pelo cliente e executa comandos)
- [http://www.lrc.ic.unicamp.br/mc833/exercicios/servidor\\_concorrente\\_2018/index.html](http://www.lrc.ic.unicamp.br/mc833/exercicios/servidor_concorrente_2018/index.html)

# Próxima aula

- Detalhes de início e finalização de conexão (melhorias nos códigos anteriores / Tratamento de sinais)