

MC833A - Programação de Redes de Computadores

Professor Nelson Fonseca

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

Roteiro

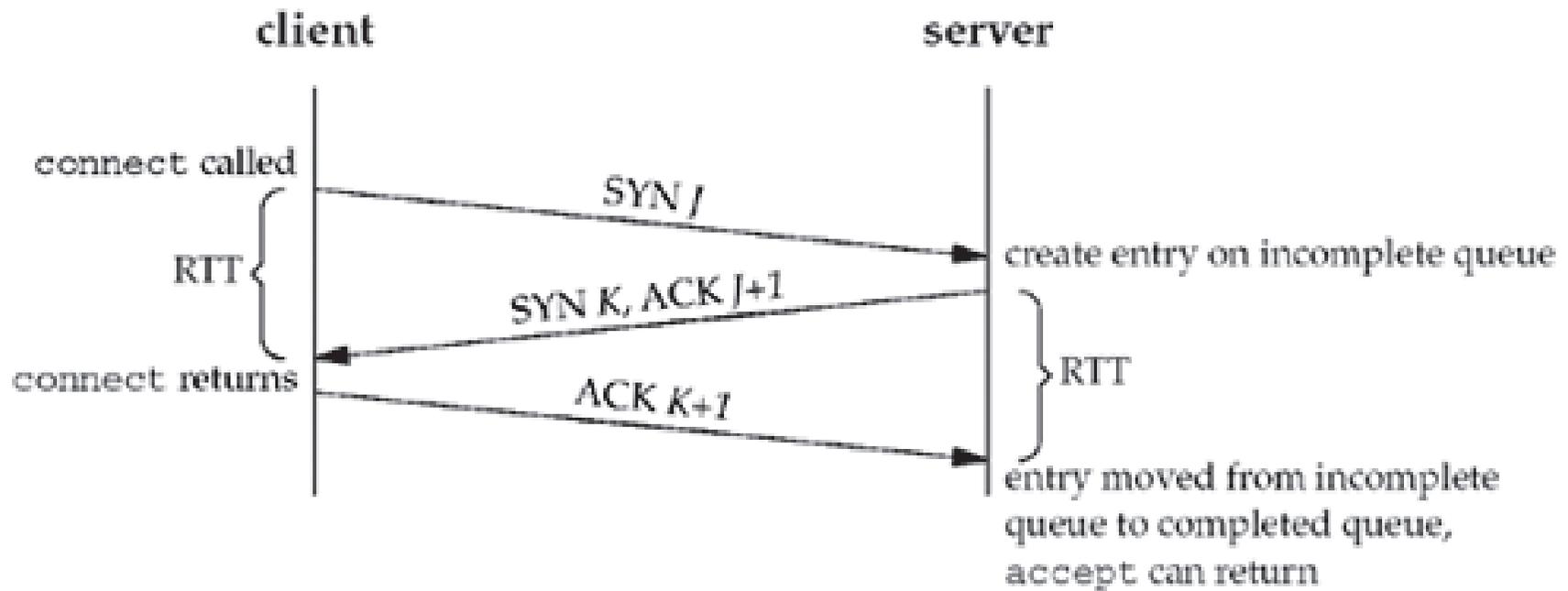
- **Objetivo: explicar o tratamento a ser feito no início e fechamento de um servidor concorrente (Capítulos 4 e 5 do livro texto)**
- `backlog` do `listen`
- Manipulando as exceções no servidor TCP concorrente
- Atividade prática

[backlog]

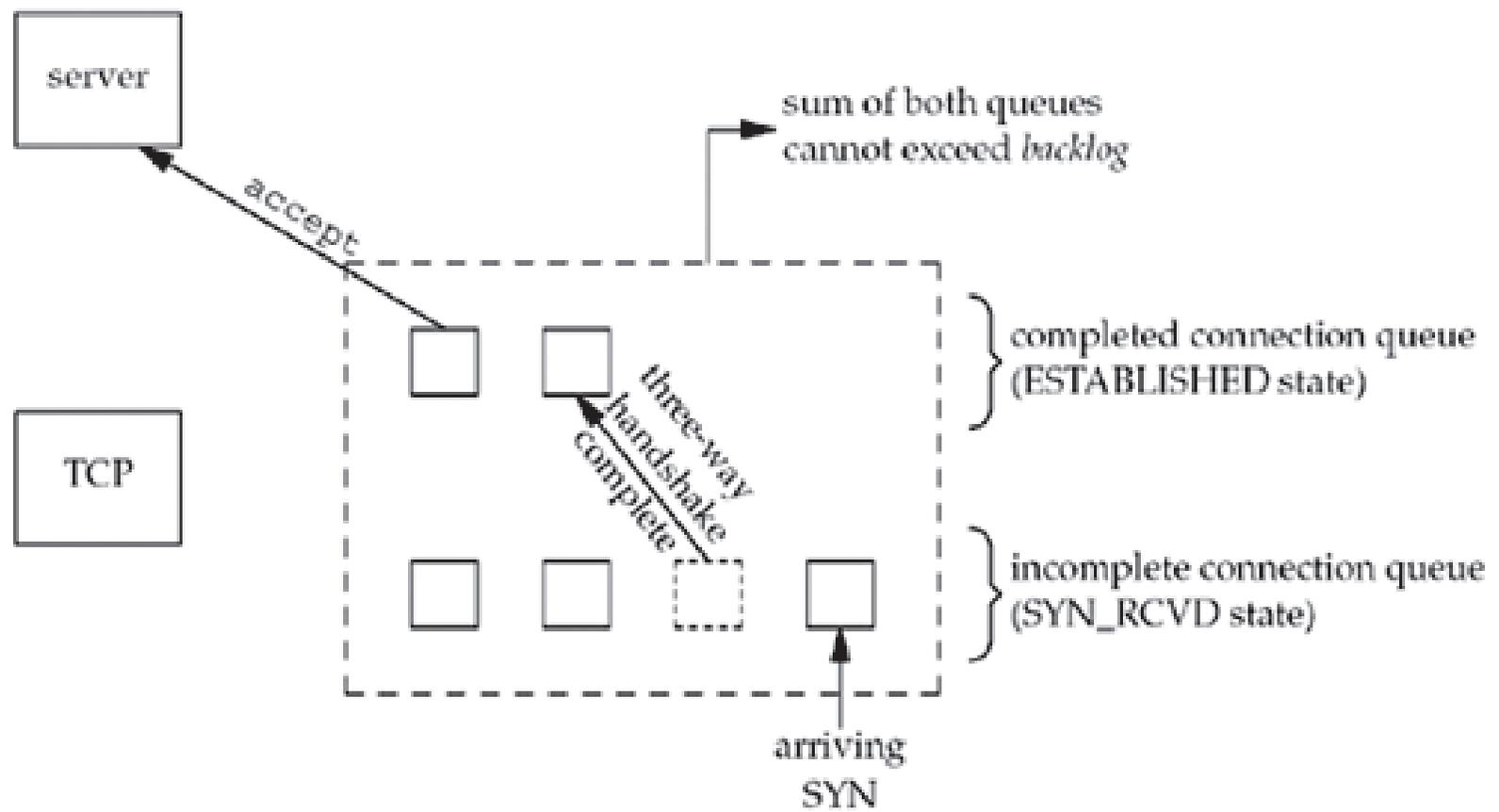
```
# int listen (int sockfd, int backlog);
```

- Move o socket do estado CLOSED para LISTEN
- backlog: Tamanho máximo das filas de conexões
- Soma de duas filas:
 - fila de conexões incompletas: SYN's que chegaram e estão esperando finalização do processo de 3-WHS (Estado SYN_RCVD)
 - fila de conexão completa: 3-WHS completo

[backlog]



[backlog]



Soma não pode exceder backlog: não é bem assim...

[backlog]

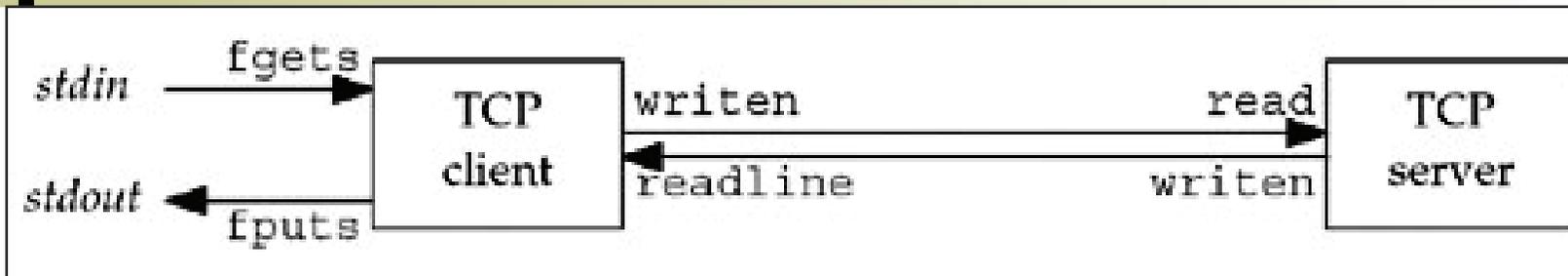
<i>backlog</i>	Maximum actual number of queued connections				
	MacOS 10.2.6 AIX 5.1	Linux 2.4.7	HP-UX 11.11	FreeBSD 4.8 FreeBSD 5.1	Solaris 2.9
0	1	3	1	1	1
1	2	4	1	2	2
2	4	5	3	3	4
3	5	6	4	4	5
4	7	7	6	5	6
5	8	8	7	6	8
6	10	9	9	7	10
7	11	10	10	8	11
8	13	11	12	9	13
9	14	12	13	10	14
10	16	13	15	11	16
11	17	14	16	12	17
12	19	15	18	13	19
13	20	16	19	14	20
14	22	17	21	15	22

- Definir no momento da execução do servidor:
 - argumento na linha de comando
 - variável de ambiente

[backlog]

- O que o servidor faz quando as filas estão cheias?
 - Ignora o SYN
 - **Por que o servidor não manda um RST finalizando a conexão?**
- O que o cliente faz quando as filas estão cheias?
 - Re-envia o SYN depois de algum tempo
 - **Qual o comportamento das aplicações cliente?**

Início e término do servidor concorrente



- A seguir veremos o que acontece...
 - quando o cliente e o servidor iniciam?
 - quando o cliente termina sua sessão com CTRL+D ou CTRL+C?
 - quando a conexão é encerrada antes do accept? (Servidor ocupado)
 - quando o cliente envia dados para um socket que “terminou”?
 - quando o servidor termina com CTRL+C?
 - quando a rede “cai” no servidor?
 - quando o servidor reinicia?
 - quando o servidor desliga?

[Servidor]

```
...
3 int main(int argc, char **argv) {
...
9  listenfd = Socket (AF_INET, SOCK_STREAM, 0);
...
14 Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
15 Listen(listenfd, LISTENQ);
16 for ( ; ; ) {
17     clilen = sizeof(cliaddr);
18     connfd = Accept(listenfd, (SA *) &cliaddr, &clilen);
19     if ( (childpid = Fork()) == 0) { /* processo filho */
20         Close(listenfd); /* fecha o que escuta */
21         str_echo(connfd); /* processa os dados */
22         exit (0);
23     }
24     Close(connfd); /* pai fecha o conectado */
25 }
26 }
```

str-echo

```
1 #include    "unp.h"
2 void
3 str_echo(int sockfd)
4 {
5     ssize_t n;
6     char    line[MAXLINE];
7     for ( ; ; ) {
8         if ( (n = Readline(sockfd, line, MAXLINE)) == 0)
9             return;          /* connection closed by other end */
10        Writen(sockfd, line, n);
11    }
12 }
```

lib/str_echo.c

Figure 5.3 str_echo function: echo lines on a socket.

[Cliente]

```
...
3 int main(int argc, char **argv) {
...
9  sockfd = Socket(AF_INET, SOCK_STREAM, 0);
...
14 Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
15 str_cli(stdin, sockfd); /* processa os dados */
16 exit(0);
17 }
```

stri-client

```
1 #include "unp.h" lib/str_cli.c  
  
2 void  
3 str_cli(FILE *fp, int sockfd)  
4 {  
5     char    sendline[MAXLINE], recvline[MAXLINE];  
  
6     while (Fgets(sendline, MAXLINE, fp) != NULL) {  
  
7         Writen(sockfd, sendline, strlen(sendline));  
  
8         if (Readline(sockfd, recvline, MAXLINE) == 0)  
9             err_quit("str_cli: server terminated prematurely");  
  
10        Fputs(recvline, stdout);  
11    }  
12 }
```

lib/str_cli.c

Figure 5.5 str_cli function: client processing loop.

[netstat (início normal)]

Lado do servidor

- **linux % netstat -a**
Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:9877	*:*	LISTEN

Lado do cliente

- **linux % netstat -a**
Active Internet connections (servers and established)

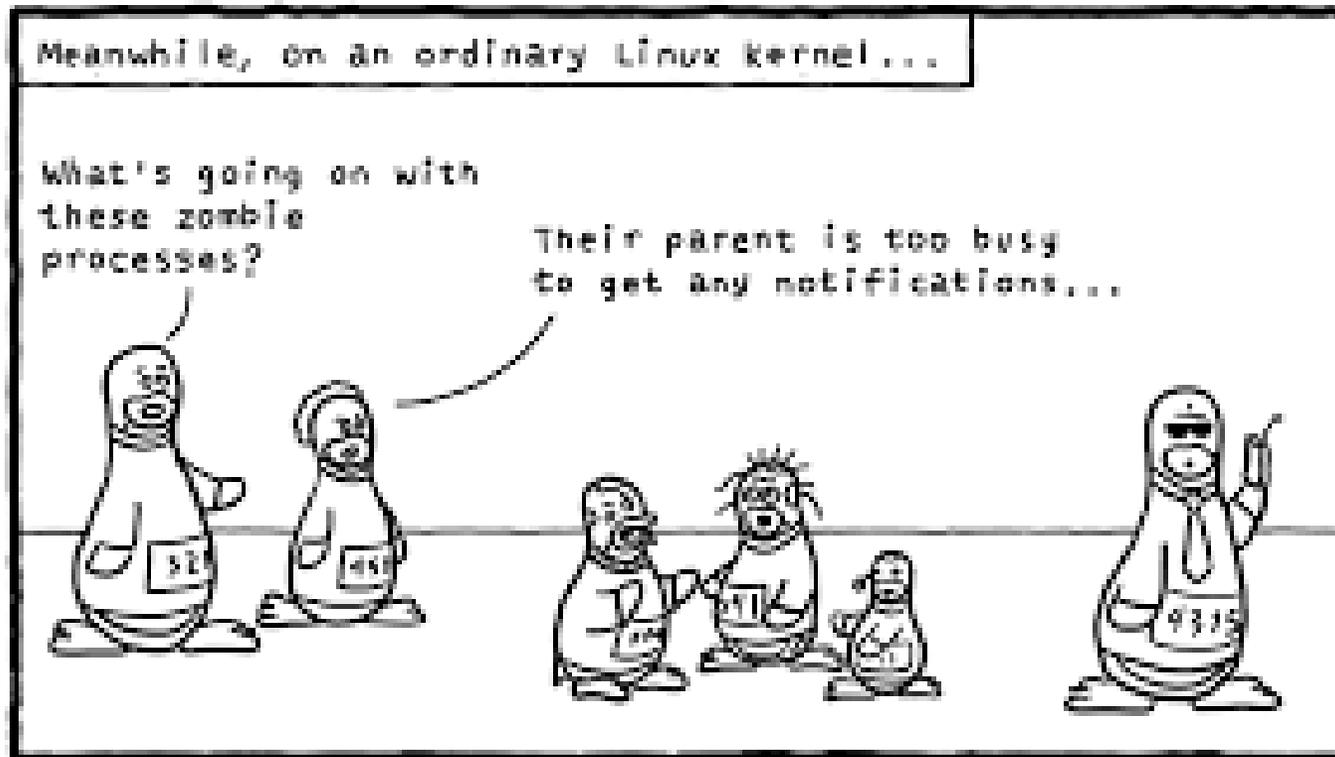
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	local host:42758	localhost:9877	ESTABLISHED

[ps (início normal)]

Processos (cliente e servidor)

■ `linux % ps -t pts/6 -o pid,ppid,TTY,stat,args,wchan`

PID	PPID	TT	STAT	COMMAND	WCHAN
22038	22036	pts/6	S	-bash	wait4
17870	22038	pts/6	S	./tcpserv01	wait_for_connect
19315	17870	pts/6	S	./tcpserv01	tcp_data_wait
19314	22038	pts/6	S	./tcpcli01 127.0	read_chan



Daniel Stern (burnsoff.us)



Algoritmo (fim normal)

1. Ao digitar EOF, `fgetc` retorna um ponteiro nulo e a função `str_cli` retorna
2. O cliente executa `exit`
3. Descriptores no cliente são fechados e um FIN é enviado ao servidor. Servidor no estado `CLOSE_WAIT` e cliente `FIN_WAIT-2`
4. O filho servidor estava bloqueado em `readline (str_echo)` e retorna para `main` servidor. Filho servidor termina executando `exit`
5. Todos os descritores do filho servidor são fechados, um FIN do servidor e um ACK são enviados ao cliente
6. Conexão terminada
7. Sinal `SIGCHLD` é enviado do processo filho para o processo pai (mas o pai não manipula o sinal... Processos zumbis)

Sinal SIGCHLD

- Sinal
 - Interrupção de software
 - Processo a processo ou kernel para processo
 - `man 7 signal`
 - `kill`, `killall`
- `sigaction` especifica *disposition* (ou *action*). Três opções para *action*:
 - Especificar o manipulador: `void handler (int signo)`
 - Ignorar, `SIG_IGN`
 - Usar o padrão, `SIG_DFL`

Código para tratar o SIGCHLD

```
2 typedef void Sigfunc(int);
3 Sigfunc * Signal (int signo, Sigfunc *func)
4 {
5     struct sigaction act, oact;
6     act.sa_handler = func;
7     sigemptyset (&act.sa_mask); /* Outros sinais não são bloqueados*/
8     act.sa_flags = 0;
9     if (signo == SIGALRM) { /* Para reiniciar chamadas interrompidas */
10 #ifdef SA_INTERRUPT
11     act.sa_flags |= SA_INTERRUPT; /* SunOS 4.x */
12 #endif
13     } else {
14 #ifdef SA_RESTART
15     act.sa_flags |= SA_RESTART; /* SVR4, 4.4BSD */
16 #endif
17     }
18     if (sigaction (signo, &act, &oact) < 0)
19         return (SIG_ERR);
20     return (oact.sa_handler);
21 }
```

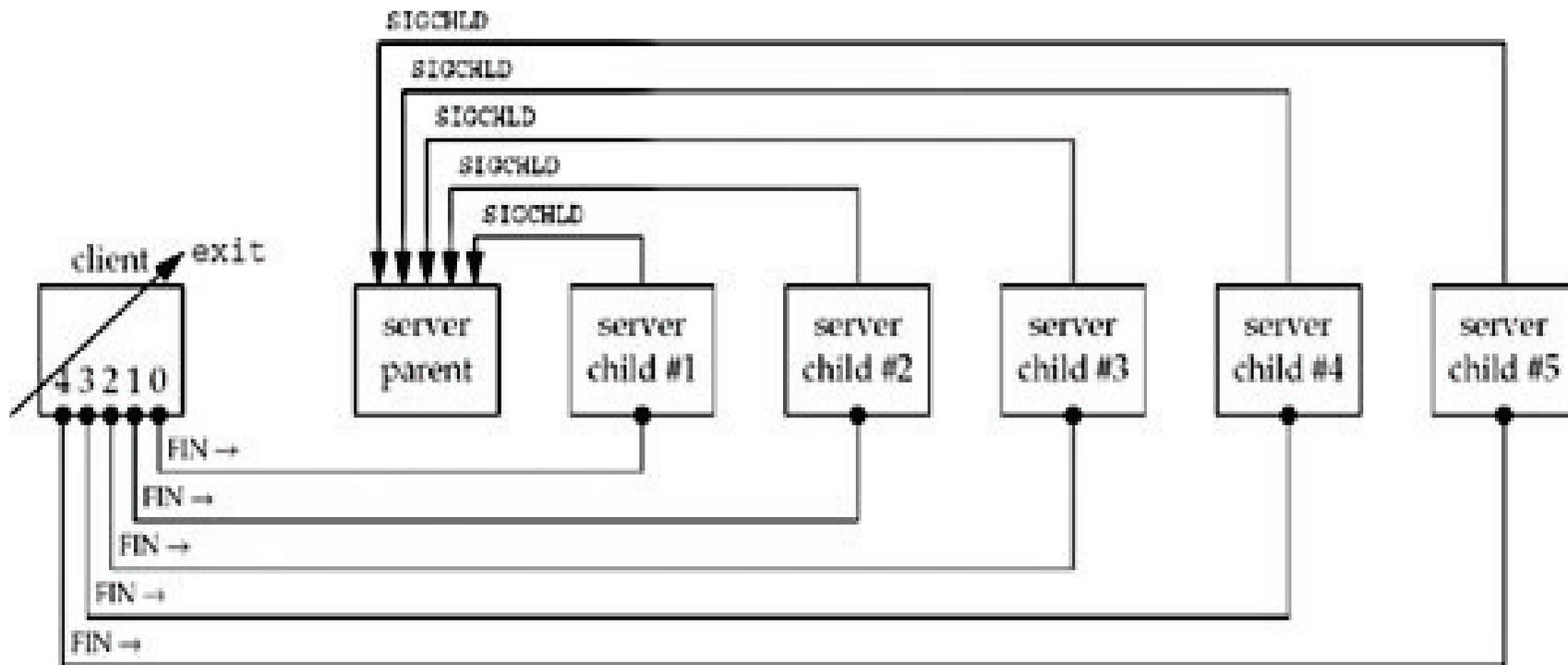
Código para tratar o SIGCHLD

- Registrar a função para o SIGCHLD 1 única vez, antes do fork.
`Signal (SIGCHLD, sig_chld)`
- Sempre que “sair” de um fork tem que esperar pelos processos filhos para evitar zumbis. Isso deverá ser feito pela função `sig_chld`.
 - `wait` ou `waitpid`

```
#include <sys/wait.h>
pid_t wait (int *statloc);
pid_t waitpid (pid_t pid, int *statloc, int options);
           Both return: process ID if OK, 0 or -1 on error
```

[Problema com wait]

- wait concorrentes → não enfileira processos zumbis



[Solução com `waitpid`]

- `Waitpid` → espera por qualquer filho terminar (primeiro argumento `-1`)
- Opção `WNOHANG` não bloqueia

```
...
2 void sig_chld(int signo) {
3     pid_t pid;
4     int stat;
5     while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
6         printf("child %d terminated\n", pid);
7     return;
8 }
```

Mudanças no servidor

- Registrar a função para manipular o SIGCHLD
- Tratar o retorno do accept caso o sinal seja recebido durante o bloqueio.

```
...
9     void sig_chld(int);
...
16    Listen(listenfd, LISTENQ);
17    Signal (SIGCHLD, sig_chld);      /* para chamar waitpid() */
18    for ( ; ; ) {
19        clilen = sizeof(cliaddr);
20        if ( (connfd = accept (listenfd, (SA *) &cliaddr, &clilen)) < 0) {
21            if (errno == EINTR)
22                continue; /* se for tratar o sinal, quando voltar dá erro
                             em funções lentas */
23            else
24                err_sys("accept error");
25        }
...

```

[Atividade prática]

- Melhorar o servidor TCP concorrente da atividade anterior (verificar os limites do `backlog` e eliminar os processos zumbis)
- <http://www.lrc.ic.unicamp.br/mc833/>

[Próxima aula]

- Multiplexação de E/S (funções `select` e `poll`)