MAC0422 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 10 de Dezembro de 2020

Roteiro

Servidores concorrentes

Lidando corretamente com os processos filho

Servidores concorrentes

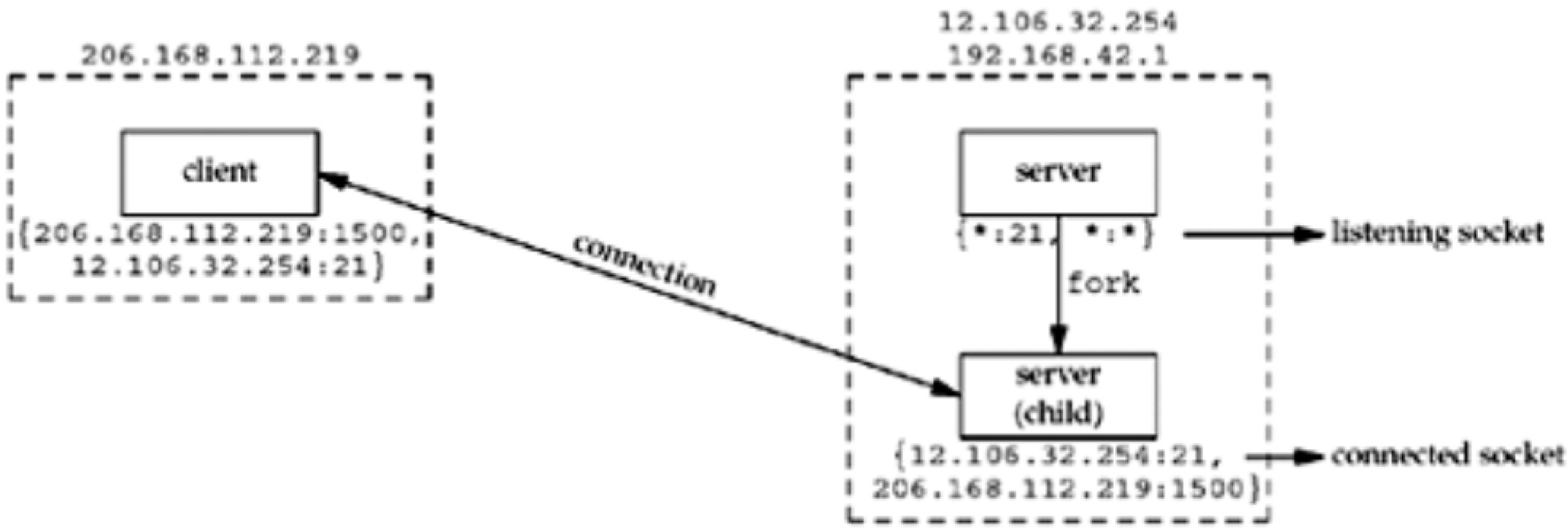
Lidando corretamente com os processos filho

Servidores concorrentes

Como criar cópias do servidor?

Servidores concorrentes

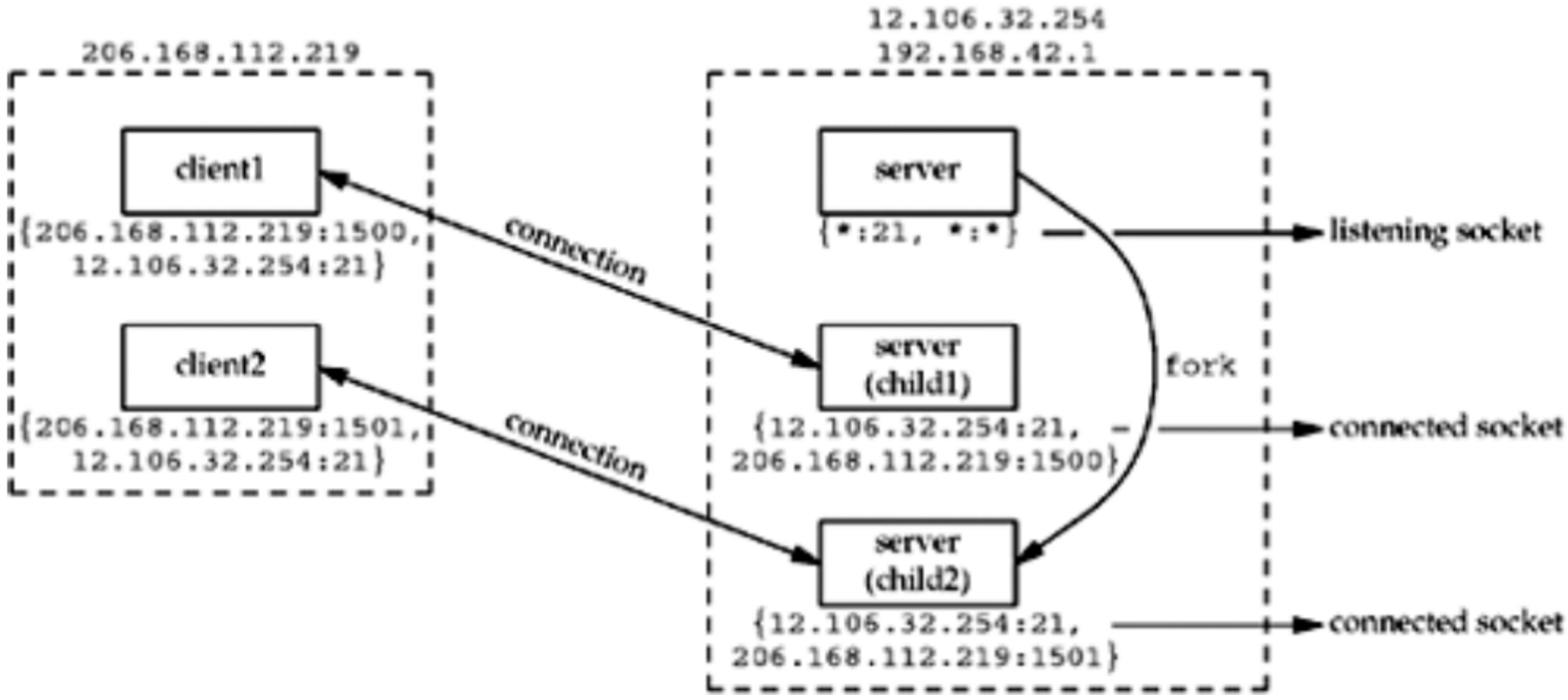
Lidando corretamente com os processos filho



Como criar cópias do servidor?

Servidores concorrentes

Lidando corretamente com os processos filho



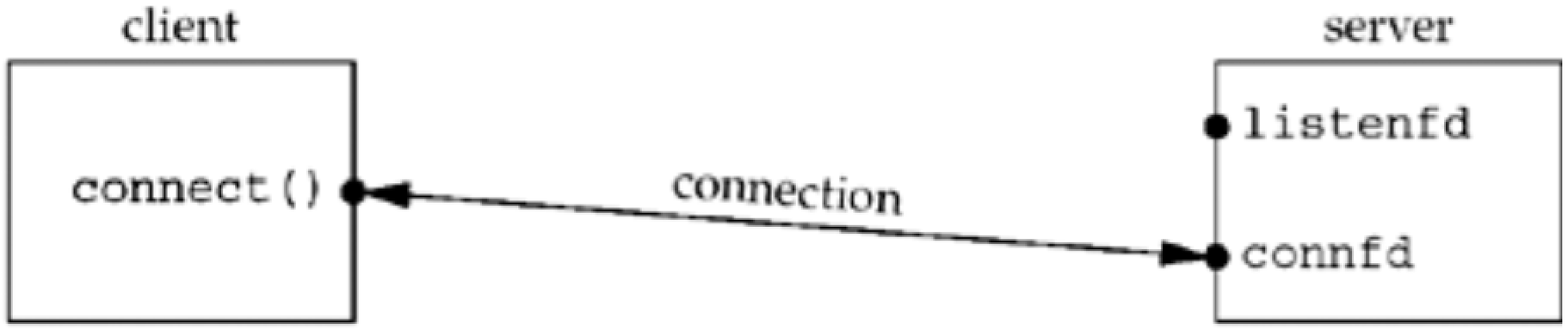
O novo servidor vai criar um novo processo


```
pid_t pid;

for ( ; ; ) {
   connfd = accept(listenfd, (struct sockaddr *) NULL,
                                                 NULL);
   if ((pid = fork()) == 0) {
      close(listenfd);
      ticks = time(NULL);
      snprintf(buff, sizeof(buff), "%.24s\r\n",
                                 ctime(&ticks));
      write(connfd, buff, strlen(buff));
      close(connfd);
      exit(0);
   }
   else
      close(connfd);
}

exit(0);
```

Mas está fechando os descritores dos sockets!

- Um arquivo / socket de fato só é fechado quando a quantidade de vezes que ele é referenciado chega a zero
- Quando fazemos fork o processo é duplicado, inclusive as referências para os arquivos / sockets abertos. Assim, para que os sockets de fato sejam fechados, é necessário fazer close em todos os processos (pai e filho)

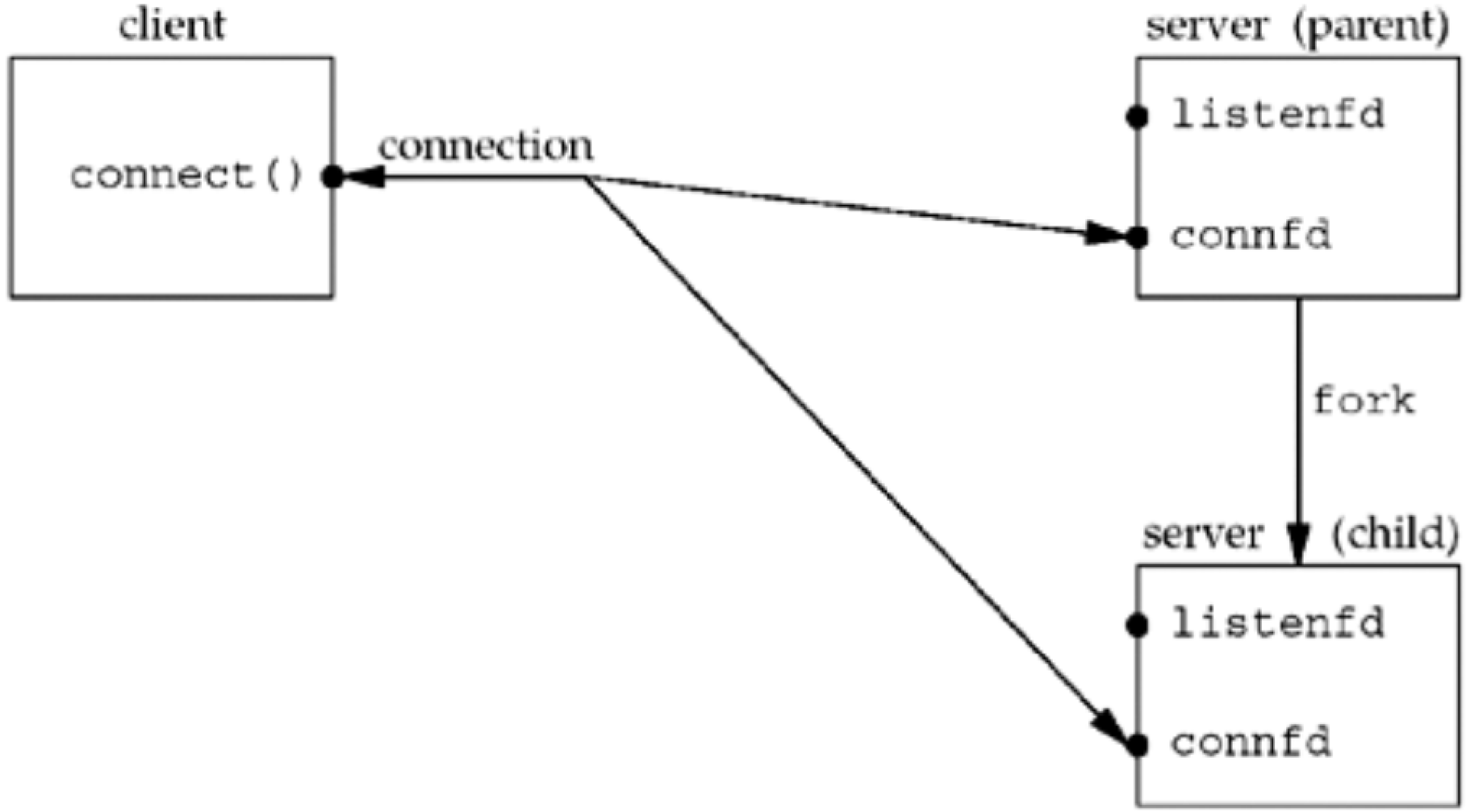
O que temos até agora?



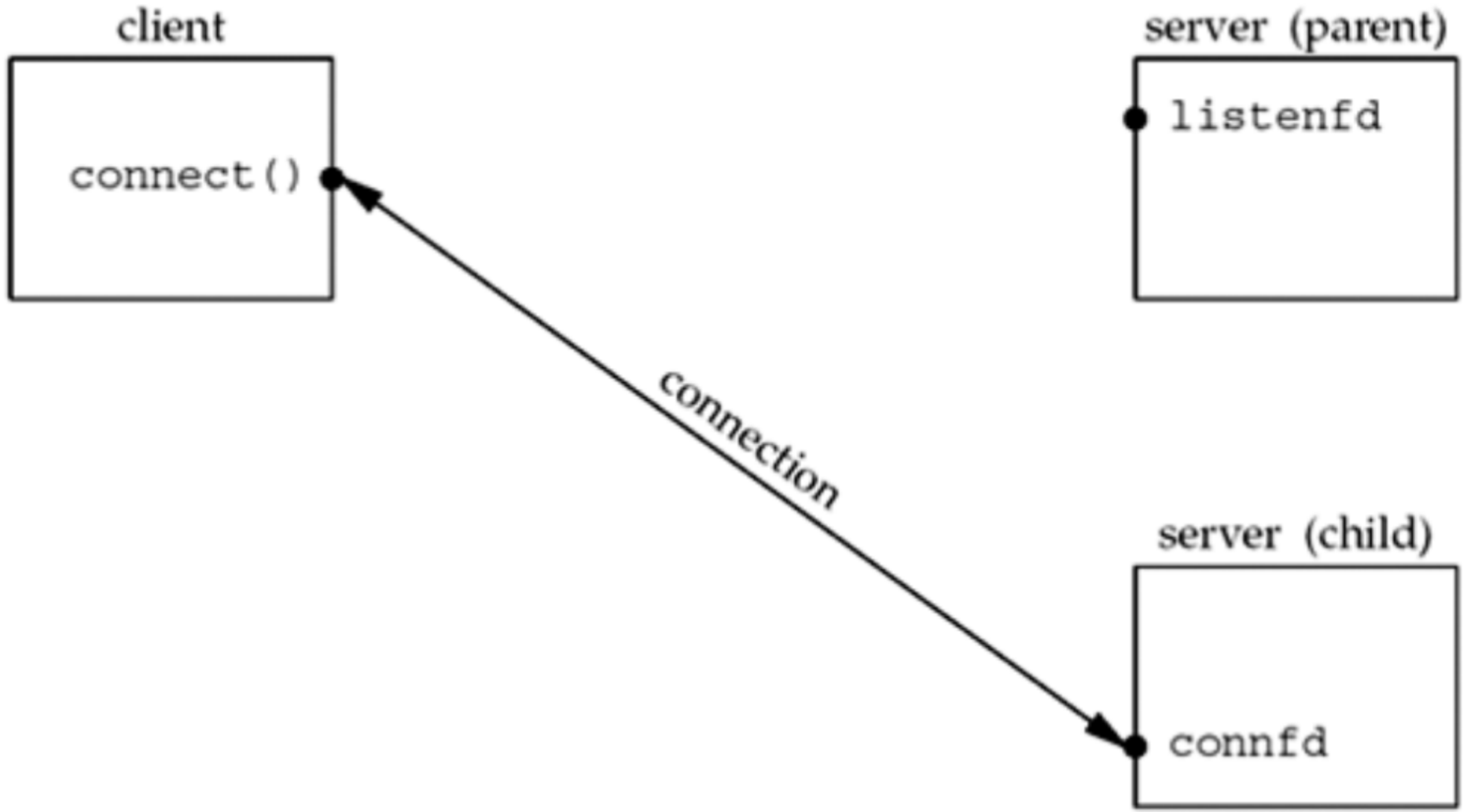
O que temos até agora?

Servidores concorrentes

Lidando corretamente com os processos filho



O que temos até agora?



Lidando corretamente com os processos filho

Como estão os processos com esse código?

Enquanto as conexões estão estabelecidas com dois clientes:

```
9965   0,0  0,0  2442000    960 s004 S+
10:33 0:00.00 ./servidor-concorrente
9963   0,0  0,0  2442000    960 s004 S+
10:33 0:00.00 ./servidor-concorrente
9957   0,0  0,0  2432780   1620 s004 S+
10:33 0:00.00 ./servidor-concorrente
```

Como estão os processos com esse código?

Servidores concorrentes

Lidando corretamente com os processos filho

Após os dois clientes finalizarem as conexões:

```
9965   0,0  0,0        0      0 s004 Z+   10:33 0:00.00 (servidor-concorr)
9957   0,0  0,0  2432780   1620 s004 S+
10:33 0:00.00 ./servidor-concorrente
```

Processos zumbi

- Sempre que um processo filho termina sua execução ele envia um sinal SIGCHLD para o processo pai.
- Por padrão a ação do pai é ignorar esse sinal. Como ele não está sendo capturado, nada acontece e o processo filho entra no estado de zumbi.
- O propósito do estado zumbi é manter informação sobre o filho para o pai capturar mais tarde. Se um processo termina e ele tem filhos zumbi, esses filhos acabam mudando de pai para o processo de PID 1 (init) que por sua vez vai limpar todos os zumbis.
- Não é uma boa ideia manter processos zumbi pois eles ficam ocupando espaço na memória do computador e desperdiçando recursos (se o processo já terminou, o ideal é removê-lo da tabela de processos)

Manipulação de sinais

- Já falamos brevemente de sinais, eles servem para um processo notificar que um evento aconteceu
- Sinais geralmente ocorrem de forma assíncrona (em momentos inesperados)
- Sinais podem ser enviados entre processos ou pelo kernel para um processo
- Todo sinal tem definido um tratamento, ou **ação** associado ao mesmo
- Para atribuir a ação, usa-se a função `sigaction`
- Uma informação importante é que sistemas baseados em Unix não enfileiram sinais se vários dos mesmos forem recebidos em um intervalo curto de tempo. Apenas um é entregue.

Manipulação de sinais

- Há 3 opções ao definir a ação com o `sigaction`
  1. Atribuir uma função que será chamada quando o signal for "capturado"
  2. Atribuir SIG_IGN para ignorar o sinal
  3. Atribuir SIG_DFL para atribuir a ação padrão quando o sinal for "capturado" (O padrão para o SIGCHLD é ser ignorado)
- Obs.: SIGKILL e SIGSTOP não podem ter suas ações ignoradas e nem modificadas

Manipulação de sinais

- Um dos argumentos da `sigaction` é uma estrutura que deve ser corretamente preenchida
- Para facilitar a vida, costuma-se chamar a função `signal` que como a própria manpage diz: "signal – simplified software signal facilities"
- O primeiro argumento do `signal` é o nome do sinal e o segundo, um ponteiro para a função que será chamada quando esse sinal for capturado (O segundo pode ser tb SIG_IGN ou SIG_DFL)

Manipulação de sinais

Servidores concorrentes

Lidando corretamente com os processos filho

```
void sig_chld(int signo) {
   pid_t pid;
   int status;

   /* WNOHANG eh para o waitpid nao bloquear caso haja
    * mais filhos rodando ainda */
   while ( (pid = waitpid(-1, &status, WNOHANG)) > 0 )
      printf("Filho %d terminou\n", pid);

   return;
}
```

Manipulação de sinais

- A atribuição da função relacionada com o sinal SIGCHLD é feita logo depois do listen:
- `signal(SIGCHLD, sig_chld);`