

20 - Sinais I

Igor Montagner

Na aula de hoje falamos sobre sinais e vimos que grande parte deles indica que algo excepcional aconteceu. Veremos neste handout os mecanismos existentes para que um processo espere seus filhos acabarem e descubra quais são as consequências da relação parental entre processos.

Parte 1 - `wait` e `kill`

Anteriormente vimos que ao chamar `wait(&status);` guardamos informações sobre o fim do processo filho na variável `status`. Nos outros exercícios olhamos para os casos em que `WIFEXITED(status) == 1`.

Todo término inesperado de um programa é feito usando um sinal. Ao acessar informações em um local de memória não mapeado para o nosso processo ele recebe o sinal `SIGSEGV`. Ao executar uma divisão por zero ele receberá o sinal `SIGFPE`. Ao apertarmos *Ctrl+C* o processo recebe o sinal `SIGTSTP` (terminal stop). Logo, nestes casos `WIFSIGNALED(status) == 1` e podemos pegar o número do sinal usando a macro `WTERMSIG(status)`.

Exercício: Crie um programa que faz um `fork()` que cria um processo filho que entra em loop infinito. O processo pai deverá esperar o fim do filho usando `wait` e usar as macros acima para mostrar o número do sinal que terminou o filho.

O envio de sinais é feito usando a chamada `kill`. Assim como outras chamadas de sistema, `kill` também é um programa de linha de comando.

Exercício: Veja a documentação da chamada de sistema (em C) no manual `man 2 kill`

Exercício: Veja a documentação da linha de comando no manual `man 1 kill`

Exercício: Uma descrição das seções do manual (o número do meio no comando acima) pode ser vista em `man man`.

Exercício: Envie o sinal `SIGINT` (Ctrl+C) para seu processo filho e verifique que o processo pai mostra o número correto.

Dica: use `man 7 signal` para uma lista de todos os sinais existentes. Teste outras combinações com `kill` e seu programa do exercício anterior.

Extra

Exercício: mostrar o número do sinal não é muito útil. Pesquise sobre a chamada `strsignal` e use-a para mostrar uma mensagem descritiva de qual sinal foi recebido.

Exercício: Abra o arquivo `fork4.c`. Modifique-o para que, após uma espera de 10 segundos, o processo pai verifique se seu filho terminou. Se ele não tiver terminado o pai envia um sinal `SIGKILL` para o filho.

Dicas: `man waitpid`, procure por `WNOHANG`.

Parte 2 - Capturando sinais

Apesar de muitos recursos mostrarem o uso da chamada `signal` para a captura de sinais, ela é considerada obsoleta e o recomendado é usar `sigaction`, que é um pouco mais complicada de usar.

O exemplo abaixo cria um `struct sigaction` e o seta para executar um handler quando o processo receber `SIGINT` (Ctrl+C).

```
void sig_handler(int num) {
    // faz algo aqui
}

....
struct sigaction s;
s.sa_handler = sig_handler; // aqui vai a função a ser executada
sigemptyset(&s.sa_mask);
s.sa_flags = 0;

sigaction(2, &s, NULL);
```

Exercício: Modifique o arquivo `signal1.c` para que o programa só termine após apertar Ctrl+C três vezes. Você pode usar `exit` para sair na terceira vez.

Para resetar o comportamento padrão de um sinal atribua a `s.sa_handler` a constante `SIG_DFL` em `s.sa_handler` e chame novamente `sigaction`.

Exercício: Restaure o comportamento original no segundo *Ctrl+C*, fazendo com que o processo realmente termine com o sinal.

Extra

Exercício: Modifique `signal1.c` para que, ao ser colocado em background usando Ctrl+Z (SIGTSTP), imprima uma mensagem antes de parar de executar.

Dicas:

- Você precisa retornar o comportamento padrão do sinal depois de dar o print.
- Pesquise como usar `raise` para (re)enviar um sinal para o próprio processo.

Exercício: Complete o programa acima com uma outra função que imprime a mensagem *Continuando!* quando o programa voltar a rodar (sinal `SIGCONT`).