ISEP – LEEC - Sistemas Computacionais 2017/2018

Ficha 3 - Sinais

Objetivos

O aluno deverá ser capaz de utilizar e aplicar os mecanismos de sinais quer do ponto de vista de utilizador que de programador. O aluno deverá ficar a conhecer os estados mais comuns de um processo.

As respostas às questões II-2.1, II-2.2, II-3.1, II-4 e II-5 e a resolução do exercício III deverão ser entregue em manuscrito e individualmente no início da próxima aula PL.

Exercício I

1) Considere o programa abaixo:

```
1:  void sigusr1(int s) {
2:    printf("Warning\n");
3:  }
4:
5:  int main() {
6:     sigset_t sigset;
7:     struct sigaction sa; sa.sa_flags = 0; sigemptyset(&(sa.sa_mask));
8:     sa.sa_handler = sigusr1;
9:     sigaction(SIGUSR1, &sa, NULL);
10:    signal(SIGTERM, SIG_IGN);
11:    signal(SIGPIPE, SIG_IGN);
12:
13:    sigemptyset(&sigset);
14:    sigaddset(&sigset, SIGPIPE);
15:    sigaddset(&sigset, SIGQUIT);
16:    sigprocmask(SIG_SETMASK, &sigset, NULL);
17:
18:    while(1){
19:     printf("Início\n");
20:     pause();
21:    }
22: }
```

Suponha que o programa é colocado em execução, sendo-lhe atribuído o identificador de processo 2000. Indique o efeito de cada um dos comandos abaixo na execução do programa

```
a) kill -QUIT 2000          b) kill -USR1 2000
c) kill -USR2 2000          d) kill -PIPE 2000
e) kill 2000                f) kill -KILL 2000
```

Exercício II

Extraia o conteúdo do ficheiro `ficha-sinais-ficheiros.zip` (disponível no moodle) para um diretório de trabalho à sua escolha. Gere os executáveis, escrevendo `make` na *shell*.

1.1) Analise o código do ficheiro `ex1.c`. Execute este programa, utilizando o comando "`./ex1`". Use a combinação **ctrl+c** para enviar o sinal SIGINT ao programa. Por *default,* quando um processo recebe este sinal, termina.

1.2) Execute este programa, utilizando o comando "`./ex1 &`" de forma a que este fique em execução em "segundo plano" (i.e., em *background).*

Processos em *foreground* e *background* – O processo que recebe diretamente os dados e comandos introduzidos pelo utilizador (tipicamente através do teclado) designa-se por processo em "*foreground*". Os restantes processos designam-se por processos em *background.*

A partir da *shell*, execute o comando `kill pid` (envio do sinal SIGTERM), onde `pid` é o identificador do processo que pretende terminar (neste caso, será o identificador do processo correspondente ao programa **ex1**). Verifique com o comando `ps` se o processo realmente terminou.

2.1) Analise o código do ficheiro `ex2.c`. Execute o programa **ex2** em *background*. Repita o procedimento da alínea 1.2. Justifique as diferenças de comportamento entre os dois programas.

2.2) Execute o comando `kill -USR1 pid`, onde `pid` é o identificador do processo correspondente ao programa **ex2**. Explique o resultado.

3.1) Execute o programa **ex3** em *background* e envie-lhe os sinais SIGTERM e SIGUSR1. Justifique as diferenças face às alíneas anteriores.

3.2) Execute o comando `kill -STOP pid`, onde *pid* é o identificador do processo. Irá observar que o processo deixa de imprimir mensagens no ecrã. Execute o comando `ps -l` e analise o resultado. Tenha em especial atenção a coluna S e a seguinte legenda (extracto do `man ps`):

Códigos do processo

```
D    Uninterruptible sleep (usually IO)
R    Running or runnable (on run queue)
S    Interruptible sleep (waiting for an event to complete)
T    Stopped, either by a job control signal or because it is being traced.
Z    Defunct ("zombie") process, terminated but not reaped by its parent.
```

3.3) Execute o comando `kill -CONT pid`, onde *pid* é o identificador do processo. Analise o resultado.

3.4) Termine o processo com o comando `kill -KILL pid`.

4) Execute o programa **ex4** em primeiro plano ("`./ex4`"). Explique como deve proceder para terminar o programa, sem terminar a *shell* (experimente as combinações **ctrl+c**, **ctrl+** e **ctrl+z**).

5) Com base na análise dos programas anteriores e nos resultados dos testes, o que conclui em relação ao comportamento dos processos face aos sinais SIGKILL e SIGSTOP?

Exercício III

1) Modifique o programa apresentado abaixo de forma a cumprir os requisitos seguintes:

- O programa deverá inicialmente bloquear os sinais SIGINT e SIGQUIT.
- Adicionalmente, os novos processos deverão ter o sinal SIGTSTP bloqueado.
- O sinal SIGTERM deverá ser ignorado em todos os processos.

A função *void myHandler(int signum)* deverá ser usada como função de atendimento para o sinal SIGCHLD no processo inicial. No entanto, esse sinal deverá voltar a ter a sua configuração por omissão (*default*) nos processos criados pelo programa.

```
#include <unistd.h>

int main() {
  while(1) {
   wait_something();
   pid_t r = fork();
   if(r == 0) {
     do_something();
     return(0);
   }
  }
  return(0);
}
```

Exercício IV

1) Quando um sinal está bloqueado, o processo tem forma de saber se esse sinal foi entretanto enviado ao processo, através da função `sigispending()`. No entanto, não tem forma de saber o número de vezes que este foi enviado. Mesmo quando o sinal não está bloqueado, pode dar-se o caso do mesmo sinal ser enviado várias vezes em rápida sucessão e o processo só detetar uma ocorrência desse sinal. Este exercício pretende ilustrar esse último fenómeno.

O programa `sigchld.c` usa o sinal SIGCHLD para detetar a terminação de processos filho. O programa foi implementado de forma a que estes terminem aproximadamente 5 segundos após a sua criação. No entanto, a implementação tem uma falha.
1.1) Execute o programa `sigchld`. Observe que, quando as impressões terminam, aparentemente ainda existem processos por terminar. No entanto, se abrir outro terminal e executar o comando "`ps -e| grep sigchld`" irá verificar que todos os processos filho já se encontram no estado *zombie* (*defunct*), isto é, já terminaram a execução do programa.
1.2) Termine o programa com a combinação **ctrl+c**. Verifique que o programa agora deteta a terminação de todos os processos filho. Analise o código e verifique de que forma esse comportamento é implementado.

1.3) Execute novamente o programa, mas desta vez prima a combinação **ctrl+c** imediatamente após a mensagem "`All processes created`". Observe que:

- A função de atendimento do SIGINT é interrompida algumas vezes pela função de atendimento do SIGCHLD.
- Os processos filho terminam devido à receção do SIGINT e não após a espera de 5 segundos.

1.4) Com base na função `int_handler`, altere a função `cld_handler` de forma que esta detete corretamente a terminação de todos os processos filho.

Apêndice

Lista de sinais e respetivas ações pré-definidas[1].

Signals described in the original POSIX.1-1990 standard:

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

Signals not in the POSIX.1-1990 standard but described in SUSv2 and POSIX.1-2001:

Signal	Value	Action	Comment
SIGBUS	10,7,10	Core	Bus error (bad memory access)
SIGPOLL		Term	Pollable event (Sys V). Synonym of SIGIO
SIGPROF	27,27,29	Term	Profiling timer expired
SIGSYS	12,-,12	Core	Bad argument to routine (SVr4)
SIGTRAP	5	Core	Trace/breakpoint trap
SIGURG	16,23,21	Ign	Urgent condition on socket (4.2BSD)
SIGVTALRM	26,26,28	Term	Virtual alarm clock (4.2BSD)
SIGXCPU	24,24,30	Core	CPU time limit exceeded (4.2BSD)
SIGXFSZ	25,25,31	Core	File size limit exceeded (4.2BSD)

```
Term - Default action is to terminate the process.
Ign  - Default action is to ignore the signal.
Core - Default action is to terminate the process and dump core (see core(5)).
Stop - Default action is to stop the process.
Cont - Default action is to continue the process if it is currently stopped.
```

[1] Extrato de `signal(7)` (“`man 7 signal`”). A versão resumida da documentação `signal(7)` está também disponível no *moodle,* em “Material adicional”.