ISEP – LEEC - Sistemas Computacionais 2023/2024

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.

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 (`sinais-iii.c`) 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 definida como função de atendimento para o sinal SIGCHLD no processo inicial, antes de ser criado qualquer novo processo. Esse sinal deverá voltar a ter a sua configuração por omissão (*default*) nos processos criados pelo programa.

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

Observação: no exame final não poderá testar o seu código num computador, portanto, nesse contexto, o funcionamento das funções `myHandler`, `wait_something` e `do_something` é irrelevante para a resolução do exercício.

Exercício IV

Altere o programa apresentado abaixo (`sinais-iv.c`), nomeadamente recorrendo ao atendimento do sinal SIGCHLD, de modo que não exista acumulação de processos *zombie* durante a sua execução.

```
void fun1(int *d) {
  ++d[0];
  printf("Novo valor gerado em fun1: %d\n", d[0]);
  sleep(2);
}

void fun2(int *d) {
  sleep(1);
  printf("Valor processado por fun2: %d\n", d[0]);
}

int main(){
  int dados = 0;

  while(1) {
    fun1(&dados);
    pid_t r = fork();
    if(r == -1) {
      perror("fork");
      exit(1);
    }
    if(r==0) {
      fun2(&dados);
      exit(0);
    }
  }
}
```

Exercício V

A função `sigispending` permite a um processo verificar se lhe foi enviado algum dos sinais cuja entrega bloqueou (`sigprocmask`). No entanto, o processo 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 modo que estes terminem aproximadamente 5 segundos após a sua criação. No entanto, a implementação tem uma falha.

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.
Conclusão: o processo pai pode não ser capaz de detetar uma ocorrência do `SIGCHLD` para cada processo filho que termina.

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.
Observação: quando se prime a combinação **`ctrl+c`** num terminal, todos os processos do grupo do processo em *foreground* (i.e., tanto o processo em *foreground* como os seus processos filho que não tenham mudado de grupo) recebem o sinal `SIGINT`

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`.
  Observação: este comportamento poderia ser evitado através da adequada configuração do campo `sa_mask` da estrutura `sigaction` e correspondente uso da função `sigaction` na configuração da função de atendimento do sinal `SIGINT`.
- Os processos filho terminam devido à receção do `SIGINT` e não após a espera de 5 segundos.

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].

Linux supports the standard signals listed below. The second column of the table indicates which standard specified the signal: "P1990" indicates that the signal is described in the original POSIX.1-1990 standard; "P2001" indicates that the signal was added in SUSv2 and POSIX.1-2001.

Signal	Standard	Action	Comment
SIGABRT	P1990	Core	Abort signal from abort(3)
SIGALRM	P1990	Term	Timer signal from alarm(2)
SIGBUS	P2001	Core	Bus error (bad memory access)
SIGCHLD	P1990	Ign	Child stopped or terminated
SIGCONT	P1990	Cont	Continue if stopped
SIGFPE	P1990	Core	Floating-point exception
SIGHUP	P1990	Term	Hangup detected on controlling terminal or death of controlling process
SIGILL	P1990	Core	Illegal Instruction
SIGINT	P1990	Term	Interrupt from keyboard
SIGKILL	P1990	Term	Kill signal
SIGPIPE	P1990	Term	Broken pipe: write to pipe with no readers; see pipe(7)
SIGPOLL	P2001	Term	Pollable event (Sys V); synonym for SIGIO
SIGPROF	P2001	Term	Profiling timer expired
SIGQUIT	P1990	Core	Quit from keyboard
SIGSEGV	P1990	Core	Invalid memory reference
SIGSTOP	P1990	Stop	Stop process
SIGTSTP	P1990	Stop	Stop typed at terminal
SIGSYS	P2001	Core	Bad system call (SVr4); see also seccomp(2)
SIGTERM	P1990	Term	Termination signal
SIGTRAP	P2001	Core	Trace/breakpoint trap
SIGTTIN	P1990	Stop	Terminal input for background process
SIGTTOU	P1990	Stop	Terminal output for background process
SIGURG	P2001	Ign	Urgent condition on socket (4.2BSD)
SIGUSR1	P1990	Term	User-defined signal 1
SIGUSR2	P1990	Term	User-defined signal 2
SIGVTALRM	P2001	Term	Virtual alarm clock (4.2BSD)
SIGXCPU	P2001	Core	CPU time limit exceeded (4.2BSD); see setrlimit(2)
SIGXFSZ	P2001	Core	File size limit exceeded (4.2BSD); see setrlimit(2)

The signals SIGKILL and SIGSTOP cannot be caught, blocked, or ignored.

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".

Histórico

- 2024-03-08 – Clarificação do enunciado do Exercício IV. Atualização do Apêndice. (jes@isep.ipp.pt)
- …
- 2007-01-01 – Versão inicial - Jorge Estrela da Silva (jes@isep.ipp.pt)