

Título: Lab02 - Laboratório sobre Processos e Sinais

- 1. Objetivo:** Apresentar uma forma de programação assíncrona; mostrar como associar sinais a processos.
- 2. Descrição sucinta/dicas para o experimento:** Sinais são formas de interação assíncronas com processos (programas em execução) e podem ser ativados através do comando kill (). O experimento a seguir demonstra o uso de sinais:
- 3. Atividades práticas:** Resolva os itens abaixo de acordo com a sequência apresentada, atentando-se para o conteúdo dos códigos e das eventuais observações que são colocadas para cada item.

a) Compile o código abaixo e veja o resultado.

```
/* teste_sinais.c ==> sinais01.c */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
int a;
void main() {
    printf("Meu pid eh %i\n", getpid());
    while(1) {
        printf("Valor de a = %i\n", a);
        a++;
        sleep(3);
    } /* fim-while */
} /* fim-main */
```

Numa outra console do Linux, execute o seguinte comando:

\$ kill -10 <número_do_processo>

Observações:

- O comando kill pode ser utilizado diretamente na console do Linux ou num código via linguagem C, por exemplo (digite man 2 kill na console do Linux para obter mais detalhes desse uso).
- Perceba que o processo é interrompido sumariamente, pois não há uma rotina de tratamento desse tipo de sinal. Portanto o programador deve escrever uma rotina de tratamento de sinais, se quiser que algo de diferente aconteça com o programa (de forma assíncrona).
- Alguns sinais tem significados especiais e uma ação padrão do sistema operacional. Por exemplo, o sinal HUP significa uma solicitação para o processo reiniciar (morrer e ressucitar imediatamente).

b) Agora altere o programa para incluir uma rotina de tratamento de interrupção como está descrito no exemplo a seguir e repita o comando descrito no passo 2.

```
/* teste_sinais.c => sinais02.c */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
int a;
/* Rotina de tratamento de sinais.*/
void tratahup(int sinal) {
    a = 0; // reinicia o valor da variavel a
} /* fim-tratahup */

void main() {
    signal (10, tratahup);
    printf("Meu pid eh %i\n", getpid());
    while(1) {
        printf("Valor de a = %i\n", a);
        a++;
        sleep(3);
    } /* fim-while */
} /* fim-main */
```

Observações:

- É possível programar sinais para que eles gerem uma determinada reação, como foi apresentado neste exemplo.
 - Esse tipo de programação é um tipo de mascaramento de um sinal pré-existente. Alguns sinais são chamados "não mascaráveis", ou seja, não é possível substituir a ação padrão, como no caso do sinal 9 - solicitação de morte para o processo.
 - Alguns sinais típicos:
 - ◆ Sinal 1: Significa reinício do programa (HUP). Este sinal é chamado de SIGHUP.
 - ◆ Sinal 2: Sinal chamado de SIGINT. Causa uma interrupção no programa, equivalente a quando se digita Control+C no teclado.
 - ◆ Sinal 15: esta é a solicitação de morte, chamada de SIGTERM. Ao receber um sinal 15 o processo deveria preparar-se para terminar, fazendo "seus últimos pedidos" e ele mesmo encerrando normalmente sua execução. Claro, isto se o programador tratar o sinal, senão será o Sistema Operacional quem o fará.
 - ◆ Sinal 9: SIGKILL. Este é a morte indefensável. Não pode ser mascarado, ou seja, o programador não consegue substituir a rotina de tratamento do Sistema Operacional que simplesmente tira o processo da fila de prontos. Deve ser usado em último caso, pois um sinal 15 é mais elegante por dar a chance ao processo de se preparar para sua morte.
 - ◆ Sinal 14: SIGALRM. Permite agendar antecipadamente o envio sinal. Vide exemplo no item a seguir.
- c) É possível criar processos que sejam insensíveis a sinais. Edite o código abaixo e faça testes para perceber este comportamento:

```
/* teste_sinais.c => sinais03.c */
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

void tratasinal(int sinal) {
    printf("\n\n***** Processo insensível a sinais ... *****\n");
    printf("***** Sinal recebido: %i \n\n", sinal);
} /* fim-tratasinal */

int main() {

    char c;
    int s;

    /* inibe todos os sinais na faixa de 1 a 34 */
    for (s=1; s<35; s++){
        signal(s, tratasinal);
    } /* fim-for */

    while (1) {
        printf("Processo ainda em execucao ... Meu pid eh %i\n", getpid());
        sleep(3);
    } /* fim-while */

    return 0;
} /* fim-main */
```

d) Teste com sinal de alarme. Edite/compile o código abaixo. Em seguida execute-o e veja os resultados de uma programação com sinal de alarme ativado.

```
/* teste_sinais.c => sinais04.c */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void trata2 (int sinal) {
    printf("Processo abortado em funcao de timeout!\n");
    raise(15);
    /* a funcao raise envia um sinal para o proprio processo.
       Esse sinal equivale a kill(getpid(), sinal). */
} /* fim-trata2 */

void trata1 (int sinal) {
    printf("Tempo esgotado... Aguardando nova digitacao dentro de 4 segundos!\n");
    /* Redefine o sinal SIGALRM para outra funcao */
    signal(SIGALRM, trata2);
    alarm(4); // agenda 4 segundos
} /* fim-trata1 */

int main() {
    int a;
    signal(SIGALRM, trata1);

    printf("Meu pid eh %i\n", getpid());
    alarm(7); // agenda 7 segundos para disparar o sinal SIGALRM
    printf("Digite um numero: \n");
    scanf("%i", &a);
    alarm(0); // cancela agendamento
    printf("Obrigado.\n");
    return 0;
} /* fim-main */
```

4. Questões a serem respondidas

- a) Faça uma pesquisa sobre todos os tipos de sinais que são factíveis entre processos.
- b) Identifique quais são os sinais emitidos quando ocorrem exceções ou operações ilegais no sistema Linux (crie programas para testar esses sinais).

Ref. bibliográficas

1. Material recolhido do site <http://www.vivaolinux.com.br/artigo/Sinais-em-Linux/>, disponível em outubro de 2010.