Sistemas Operacionais I

Sinais no Linux

Sinais (*signals*)

- Interrupções de software
- Maneira de trabalhar com eventos assíncronos
- Exemplo:
  - quando um processo termina (normal ou anormalmente) envia o sinal SIGCHLD ao seu pai.
  - pai pode ignorar o sinal (padrão) ou captura-lo com a função wait

Sinais (*signals*)

- Indicam eventos inesperados ou imprevisíveis
- Cada sinal tem um nome: SIG + …
- Exemplos:
  - SIGABRT é gerado quando um processo chama a função abort
  - SIGALRM é gerado quando espira um temporizador criado pela função alarm

Sinais (*signals*)

- Condições que geram sinais:
  - Combinações de teclas (Ex.: Control+C gera SIGINT)
  - Exceções de Hardware (Ex.: divisão por zero gera SIGFPE)
  - Condições de Software (Ex.: término do temporizador criado com função alarm gera SIGALRM)
  - Chamada a função kill em C, que envia um sinal para outro processo ou chamada ao comando kill no Unix que é uma interface à função kill (Ex.: digitar no shell o comando kill -USR1 1234 envia SIGUSR1 ao processo de pid 1234)

Tratamento de Sinais

- Quando um sinal ocorre é possível associar uma ação (função):
  - Ignorar o sinal: funciona para quase todos os sinais com exceção de SIGKILL e SIGSTOP
  - Capturar o sinal: informar o núcleo para chamar uma função quando um sinal ocorrer
  - Deixar a ação padrão acontecer: cada sinal possui uma ação padrão associada

Alguns Sinais e suas Ações

Nome	Descrição	Ação Padrão
SIGABRT	término anormal (abort)	terminar + core
SIGALRM	espirou um temporizador (alarm)	terminar
SIGCHLD	mudou o status de um processo filho	ignorar
SIGCONT	continua processo parado	continuar / ignorar
SIGFPE	exceção aritmética	terminar + core
SIGINT	caractere de interrupção do terminal (Control-C)	terminar
SIGKILL	terminação. Não pode ser capturada / ignorada	terminar
SIGPIPE	escrita para pipe sem leitores	terminar
SIGSTOP	pára um processo	pára processo
SIGTERM	terminação (padrão do kill).	terminar
SIGUSR1	sinal definido pelo usuário	terminar
SIGUSR2	sinal definido pelo usuário	terminar

Função Signal

- Permite que um processo especifique que ação deve ser tomada quando determinado sinal ocorre
- Chamada:

void (*signal (int sigCode, void (*func) (int))) (int);

  - sigCode especifica o sinal
  - func pode ser:
    - SIG_IGN: sinal especificado deve ser ignorado
    - SIG_DFL: sinal deve ser tratado com ação padrão
    - uma função definida pelo usuário
  - Retorno é a função / valor previamente associado ou -1 em caso de erro

Função Signal

- Um processo herda as configurações de sinais do pai em um fork
- Entretanto, com exec as configuração de sinais são resetadas
- Com exceção de SIGCHLD, sinais não são empilhados

Exemplo com signal

```
#include <signal.h>
#include <stdio.h>

static void sig_usr(int); /*função que gerencia os dois sinais */

int main(void)
{
        if (signal(SIGUSR1, sig_usr) == SIG_ERR)
                printf("Nao pode capturar SIGUSR1\n");
        if (signal(SIGUSR2, sig_usr) == SIG_ERR)
                printf("Nao pode capturar SIGUSR2\n");
        for ( ; ; )
;
}
static void sig_usr(int signo) /* argumento e' numero do sinal */
{
        if (signo == SIGUSR1)
                printf("recebido SIGUSR1\n");
        else if (signo == SIGUSR2)
                printf("recebido SIGUSR2\n");
        else
                printf("recebido sinal %d\n", signo);
        return;
}
```


Exemplo com Signal

- Comandos no shell:

$ gcc -o sigexemp sigexemp.c
$ ./sigexemp &
[1] 536
$ kill -USR1 536
recebido SIGUSR1
$ kill -USR2 536
recebido SIGUSR2
$ kill 536
[1]+ Terminated ./sigexemp

Função Kill

- Envia um sinal a um processo
- Chamada:

int kill (pid_t pid, int sigCode);

  - sigCode especifica o sinal
  - pid especifica o número do processo para envio
  - Processo que recebe o sinal deve ser do mesmo usuário que envia ou quem envia é super-usuário
  - Retorna 0 caso consiga enviar pelo menos um sinal com sucesso e -1 em caso de erro

Função raise

- Envia um sinal ao próprio processo chamador
- Chamada:

  int raise (int sigCode);

  - sigCode especifica o sinal
  - Retorna 0 caso consiga enviar o sinal com sucesso e -1 em caso de erro
  - Equivale a kill(getpid(), sigCode)

Função alarm

- Permite determinar um temporizador que irá espirar após um tempo específico
- Quando espira SIGALRM é gerado
- Se não for capturado ou ignorado, a ação padrão é terminar o processo
- Chamada:
- unsigned int alarm (unsigned int sec);
  - sec especifica o número de segundos
  - Retorna 0 ou o número de segundos até espirar alarme prévio

Função alarm

- Se alarme já havia sido solicitado, anterior é sobrescrito
- Se argumento passado for zero, cancela todos os alarmes pendentes
- Exemplo:

```
#include <stdio.h>
#include <unistd.h>
int main (void)
{
   alarm (5); /*O alarme vai desparar daqui a 5 segundos*/
   printf("O processo é eterno, enquanto dura...\n");
   while (1);
   printf("Linha que nunca aparece...\n");
}
```

Função Pause

- Suspende o processo chamador até que um sinal seja capturado
- Chamada:

  int pause (void);
- Retorna -1 caso um manipulador de sinal seja executado
- espera eficiente por um sinal ao invés de espera ocupada

Exercícios

1) Faça um programa que trate o recebimento do sinal SIGUSR1. Quando esse sinal for recebido, o programa deve mostrar seu pid e ppid na tela.
2) Faça um programa que envie um sinal SIGUSR1 ao processo anterior. Depois de 5 segundos ele manda um SIGKILL.
3) Desenvolva um programa que executa uma repetição até que o usuário digite o número 10. O programa deve ignorar o Control-C.

Exercícios

4) Faça um programa que cria um processo filho. O filho executará o programa que foi passado para o pai como segundo argumento. O primeiro argumento do pai especifica quanto tempo o filho pode executar no máximo. Utilizando alarme, faça o filho terminar quando esgotar o tempo e mostre a mensagem "Filho esgotou o tempo!".

5) Desenvolva um programa que trate dois sinais a sua escolha. Gere os sinais, na linha de comando, e teste o programa.