Sinais no Unix

Laboratório de Sistemas Operacionais

Prof. MSc. João Tavares

Sinais

- Sinais → Interrupções de software

- Maneira de trabalhar com eventos assíncronos
  - Indicam eventos inesperados ou imprevisíveis que precisam ser tratados pelo processo

- Cada sinal tem um nome SIG+...

Ex.:

– Quando um processo termina, o SO envia o sinal SIGCHLD ao seu processo pai

– O processo pai pode ignorar o sinal (padrão) ou tratá-lo (ex.: com a função wait())

Condições que geram os sinais

- Exceções de Hardware
  - divisão por zero;
  - acessos inválidos à memória; etc.

- Condições de Software controladas pelo SO
  - término do temporizador criado com função alarm();
  - término de processo filho; etc.

- Requisição explícita de outro processo
  - processo usa função kill() em C
  - execução do comando kill ou combinações especiais de teclas no shell

Gerando sinais no Shell

- O comando kill envia sinal a um processo qualquer
  – Pode ser usado no shell pelo usuário ou em um script
  Ex.: envia sinal SIGUSR1 ao processo de pid 1234
  $ kill -USR1 1234

- Combinações especiais de teclas geram sinais enviados ao “processo” que está em primeiro plano
  - Driver do terminal (tty) captura tecla e mapeia para sinal enviado ao processo (job)
  Ex.:
  Ctrl+C → envia SIGINT
  Ctrl+\ → envia SIGQUIT
  Ctrl+z → envia SIGTSTP (congela processo)

Entrega de sinal a um processo

- Sinal enviado a um processo, caso não esteja bloqueado, é anotado (no BCP) como pendente – E se estiver bloqueado?

- Em algum momento o SO dispara o tratamento
  1) Execução do processo é congelada
  2) Contexto atual de execução é salvo
  3) **Ação configurada para o sinal pendente é executada no contexto do processo**
  4) Contexto original de execução é restaurado
  5) Execução do processo é retomada
     - Se processo dormindo em uma chamada bloqueante, chamada é tipicamente abortada retornando EINTR

Registro de sinais no BCP

- Máscara de sinais
  – Registra que sinais estão bloqueados pelo processo
  – A entrega de um sinal bloqueado é atrasada pelo SO até que o mesmo seja desbloqueado pelo processo
  – SIGKILL e SIGSTOP não podem ser bloqueados!

- Lista de sinais pendentes
  – Registra sinais que foram entregues ao processo, mas para os quais o tratador ainda não foi executado
  – Para sinais não enfileiráveis, haverá no máximo um sinal pendente daquele tipo

Tratamento de sinais

- Tratar sinal → programar reação ao sinal
  – Definir que ação será tomada pelo processo quando cada tipo de sinal chegar
  – Se omitido, ação padrão (pré-definida) será adotada
  – Ação atualmente configurada também é chamada de disposição do sinal

- Não funciona para **SIGKILL** e **SIGSTOP**!
  – Esses dois sinais são gerenciados no nível do SO
  – Não chegam efetivamente ao processo

Tratamento de sinais

- Comportamentos (disposições) possíveis
  – Ignorar
    * Ao ser entregue, o sinal será descartado

  – Capturar
    * Processo registra uma função para ser chamada quando sinal chegar

  – Ação padrão
    * Deixar a ação padrão, pré-definida no sistema para cada tipo de sinal, acontecer
    * Tipicamente, ou o sinal é ignorado ou causa o término do processo

Sinais POSIX

- O padrão POSIX define duas categorias de sinais confiáveis com caraterísticas distintas

- Sinais convencionais
  – Não são enfileiráveis → múltiplas ocorrências próximas combinadas em uma única entrega
  – Tem significado/ações-padrão pré-definidos

- Sinais RT (Tempo Real)
  – Podem ser enfileirados e ter um parâmetro
  – São entregues em ordem bem definida
  – Não há significado pré-definido para cada sinal
    * Ação padrão é sempre terminar o processo

Ações padrão para alguns Sinais

Sinal	Descrição	Ação Padrão
SIGABRT	Término anormal, gerado por abort()	Terminar+Core
SIGALRM	Expirou o temporizador programado com alarm()	Terminar
SIGCHLD	Mudou o estado de um processo filho.	Ignorar
SIGCONT	Retoma execução de processo se parado	Continuar
SIGFPE	Exceção aritmética	Terminar+Core
SIGINT	Interrupção gerada pelo terminal (Ctrl+C)	Terminar
SIGKILL	Termina processo imediatamente.	Terminar
SIGPIPE	Tentativa de escrita para pipe sem leitores.	Terminar
SIGSTOP	Suspende execução do processo até SIGCONT	Parar
SIGTERM	Termina processo (pode ser capturado).	Terminar
SIGUSR1	Sinal definido pelo usuário	Terminar
SIGUSR2	Sinal definido pelo usuário	Terminar

- Para a lista completa → *man 7 signal*

Alterando a disposição de Sinais

- Diversas APIs disponíveis, diferentes semânticas
  – ANSI-C → signal()
  – SystemV → sigset(), sysv_signal()
  – BSD → sigvec(), bsd_signal()
  – POSIX → sigaction()

- API ANSI-C, em particular, tem grandes variações de implementação.

- **Portabilidade** → preferir POSIX em programas novos

Alterando a disposição de Sinais

API

```
int sigaction( int signum,
               const struct sigaction *act,
               struct sigaction *oldact)
```

- Descrição
  – Define nova disposição (tratamento) do sinal *signum* para aquela definida na estrutura *act*
    * **Não** recebe o sinal na chamada!
    * Configuração antiga opcionalmente copiada em *oldact*

- Campos da struct *sigaction*
  sa_handler → tratador de sinal convencional
  sa_sigaction → tratador de sinal estendido (RT)
  sa_mask → máscara de sinais bloqueados
  sa_flags → modificam tratamento do sinal

Alterando a disposição de Sinais

- Campo sa_handler armazena ponteiro para função que tratará o sinal
  – Qualquer função compatível com "*void func(int)*"
    * Nome da função não é relevante
    * Tem um único parâmetro do tipo int que é o número do sinal recebido
    * Não retorna nenhum valor

  Ex.:

  void meu_tratador_sinal(int signum) {…}

- Constantes pré-definidas para sa_handler
  SIG_DFL → função que implementa a ação padrão
  SIG_IGN → função que descarta o sinal

Exemplo com sigaction()

```c
/* Includes omitidos, consulte as man pages */

void tratador(int signum) {
  printf("Recebido sinal %d\n", signum);
}

int main() {
  struct sigaction sa;
  memset(&sa, 0, sizeof(sa)); /* inicializa com zeros */

  sa.sa_handler = &tratador;

  printf("Meu PID=%d\n", getpid());

  if (sigaction(SIGUSR1, &sa, NULL) != 0) {
     perror("Falha ao instalar tratador de sinal");
     exit(-1);
  }
  for(;;) {} /* espera (busy-waiting) pelo sinal */
  return 0;
}
```

Exemplo com sigaction()

1) Compilar e executar o programa em um terminal

Ex.:

```
$ gcc ex-sigaction.c -o ex-sigaction
$ ./ex-sigaction
Meu pid e 1234
```

2) Em outro terminal, utilizar o comando kill para enviar sinais ao processo

Ex.:

```
$ kill -USR1 1234
$ kill -TERM 1234
```

3) Observar efeito dos sinais no processo que está no primeiro terminal

Semântica de sinais x fork()/exec()

- Durante um fork()...
  – Disposições de sinais são herdadas pelo filho
  – Máscara de sinais bloqueados também é herdada
  – Lista de sinais pendentes é limpa

- Durante um exec()...
  – As disposições de sinais são ajustadas
  * Para sinais capturados são restauradas as ações padrão
  * Para sinais ignorados são mantidas inalteradas
  – A máscara de sinais bloqueados e lista de sinais pendentes é preservada

Enviando Sinais

API

```
int kill( pid_t pid, int sig )

int raise( int sig )
```

- Descrição
  – kill() → envia sinal ao processo identificado pelo *pid*
  – raise() → envia o sinal especificado ao processo ou thread chamadora
  – Retornam -1 caso não consigam enviar o sinal

- Situações de falha do kill()
  – Processo destino não existe
  – Processo chamador não tem permissões adequadas
    * Credencial não é mesma do processo alvo, nem *root*

Agendando Alarmes

API

```
unsigned int alarm(unsigned int seconds)
```

- Descrição
  – Agenda o envio de um sinal SIGALRM ao processo chamador após o número especificado de segundos
  * Substitui qualquer agendamento antigo
  * Cada processo tem um único alarme agendado por vez
  * Não espera a entrega do sinal!

  – Retorna:
  * número de segundos remanescentes até o próximo alarme que foi agendado; ou
  * zero se não existe alarme agendado

- Se SIGALRM não for tratado, processo é terminado!

Exemplo com alarm()

- ex-alarm.c

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
   /* agenda SIGALRM para daqui a 5 segundos */
   if ( alarm(5) < 0 ) {
      perror("Falha ao agendar alarme");
   }
   printf("O processo é eterno, enquanto dura...\n");
   /* espera ocupada (busy-waiting) pelo alarme */
   while (1);

   printf("Linha que nunca aparece...\n");
   return 0;
}
```

Evite isso em seus programas!

- O que acontece ao processo quando o SIGALRM é entregue? Por quê?

Espera eficiente por Sinais

API

```
int pause(void)

int sigsuspend(const sigset_t *mask)
```

- Descrição
  – pause() → suspende até recepção de sinal qualquer
  – sigsuspend() → suspende e aguarda sinais específicos
  * Sinais aguardados estão ausentes em *mask*
  * Sinais em *mask* são temporariamente bloqueados
  * sigset_t → tipo para conjunto de sinais (padrão POSIX)
  = Funções relacionadas: sigemptyset(), sigfillset(), sigaddset(), sigdelset(), sigismember()

- Tratador do sinal é sempre executado antes de pause() ou sigsuspend() retornar
  – Disposição do sinal não pode ser SIG_IGN!

Exemplo com sigsuspend()

- *ex-sigsuspend.c*

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>

/* var. compartilhada entre main() e tratador de sinal */
volatile sig_atomic_t counter = 0;

/* tratador de sinal*/
void count_signal(int signum) {
   counter++;
}
```

- Continua no próximo slide...

Exemplo com sigsuspend()

```
int main() {
  sigset_t  mask;
  struct sigaction action;
  memset(&action, 0, sizeof(action));
  action.sa_handler = &count_signal;

  if (sigaction(SIGUSR1, &action, NULL) == -1) {
    perror("Falha em sigaction"); exit(-1);
  }
  printf("Meu PID=%d\nEnvie SIGUSR1 para continuar, ou"
         "SIGINT para sair\n", getpid());

  /* seleciona todos os sinais exceto SIGINT e SIGUSR1*/
  sigfillset(&mask);
  sigdelset(&mask, SIGUSR1);
  sigdelset(&mask, SIGINT);
  while (1) {  /* loop aguardando chegada de sinal */
    sigsuspend(&mask); printf("Contador=%d\n", counter);
  }
  return 0;
}
```

Outras informações

- Antiga API de sinais baseada em signal() tornou-se obsoleta pelo padrão POSIX-2001

- Outras funções definidas nesse padrão incluem:
  – sigqueue() → alternativa a kill() para enviar sinais
  * Permite enviar parâmetro juntamente com o sinal
  * Parâmetro somente será recebido caso o processo tenha instalado um tratador de sinal estendido

  – sigprocmask() → permite examinar e modificar a máscara de sinais (bloqueados) pelo processo
  * Sinais bloqueados não são descartados!
  * Apenas a entrega (ativação do tratador) é postergada

  – sigpending() → retorna lista de sinais pendentes

Leituras complementares

- STEVENS, W.R. Advanced Programming in the UNIX Environment. 2nd. Ed., Addison Wesley, 2005.

- Man pages
  – cada uma das funções abordadas
  – signals(7) → provê visão geral da API

- Página info da libc, em especial as seções
  - Em particular a seção “Signal Handling”

- Livro: Advanced Linux Programming
  Disponível para download em:
  http://www.advancedlinuxprogramming.com/alp-folder

Referências Bibliográficas

- Material originalmente elaborado por Prof. Cristiano Costa. Material autorizado e cedido pelo autor. Revisado e atualizado por Prof. Luciano Cavalheiro e posteriormente pelo Prof. João Tavares.