Laboratório de Pesquisa em Redes e Multimídia

IPC no UNIX

Sinais

Universidade Federal do Espírito Santo
Departamento de Informática

Introdução – Sinais (1)

- Uma forma de comunicar a ocorrência de eventos para os processos são os sinais.
- Os sinais são gerados quando o evento ocorre pela primeira vez e são considerados entregues quando o processo reage ao sinal.
- Os sinais são considerados como comunicação assíncrona.
- Origens
  - Hardware (e.g. SIGSEGV)
  - Kernel (*SIGIO*)
  - Processos (alarm(), kill ())
  - Usuário (e.g. control-c --> SIGINT)

Introdução – Sinais (2)

- Sinais vêm desde Unix original
- Portabilidade
  - O padrão POSIX 1003.1 define a interface, mas não regulariza a implementação
  - SVR2: mecanismo pouco confiável (defeituoso) de notificação de sinais
  - 4.3BSD: mecanismo robusto, mas incompatível com a interface SV
  - SVR4:
    - compatível com POSIX
    - incorporou várias características do BSD;
    - compatível com versões mais antigas de SV
- São caros porque emissor tem que fazer syscall e kernel tem que mexer na pilha do receptor.
- Banda limitada: apenas 31 em SVR4 e 4.3BSD, 64 em AIX, Linux.

Tipos de sinal (1)

- Os sinais são identificados pelo sistema por um número inteiro.
- Cada sinal é caracterizado por um mneumônico
- O arquivo /usr/include/signal.h contém a lista de sinais acessíveis ao usuário
- A maioria dos SOs Unix fornecem dois sinais de usuários (genéricos) para um processo usar como quiser:
  - SIGUSR1 - sinal de usuário 1;
  - SIGUSR2 - sinal de usuário 2.

Tipos de sinal (2)

- Alguns exemplos de sinais usados nas aplicações em UNIX
  - SIGINT
    - Interrupção: sinal emitido aos processos do terminal quando as teclas de interrupção (INTR ou CTRL-c) do teclado são acionadas.
  - SIGKILL
    - Destruição: arma absoluta para matar os processos. Não pode ser ignorada, nem interceptada (veja SIGTERM para uma morte mais suave para processos)
  - SIGTERM
    - Terminação por software: emitido quando o processo termina de maneira normal. Pode ainda ser utilizado quando o sistema quer por fim à execução de todos os processos ativos.
  - SIGSEGV
    - Emitido em caso de violação da segmentação: tentativa de acesso a um dado fora do domínio de endereçamento do processo.
  - SIGCHLD
    - Morte de um filho: enviado ao pai pela terminção de um processo filho
  - SIGALRM
    - usado pela chamada de sistema alarm(...);

Tipos de sinal (3)

Sinais definidos no POSIX

Signal	Description	default action
SIGABRT	process abort	implementation dependent
SIGALRM	alarm clock	abnormal termination
SIGBUS	access undefined part of memory object	implementation dependent
SIGCHLD	child terminated, stopped or continued	ignore
SIGCONT	execution continued if stopped	continue
SIGFPE	error in arithmetic operation as in division by zero	implementation dependent
SIGHUP	hang-up (death) on controlling terminal (process)	abnormal termination
SIGILL	invalid hardware instruction	implementation dependent
SIGINT	interactive attention signal (usually ctrl-C)	abnormal termination
SIGKILL	terminated (cannot be caught or ignored)	abnormal termination
SIGPIPE	write on a pipe with no readers	abnormal termination
SIGQUIT	interactive termination: core dump (usually ctrl-\)	implementation dependent
SIGSEGV	invalid memory reference	implementation dependent
SIGSTOP	execution stopped (cannot be caught or ignored)	stop
SIGTERM	termination	abnormal termination
SIGTSTP	terminal stop	stop
SIGTTIN	background process attempting to read	stop
SIGTTOU	background process attempting to write	stop
SIGURG	high bandwidth data available at a socket	ignore
SIGUSR1	user-defined signal 1	abnormal termination
SIGUSR2	user-defined signal 2	abnormal termination

Processamento de Sinais (1)

- Três fases para processar sinais:
  - **Geração**: algum evento ocorre exigindo que um processo seja notificado do mesmo
  - **Pendência (pending)**: entre a geração e a entrega
  - **Entrega** (delivery): o processo reconhece o sinal e toma a ação apropriada (**tratamento**)

Geração de Sinais (1)

- Exceções
  - kernel notifica o processo com um signal
- IPC
  - uso de kill() ou raise()
- Interrupção via Terminal
  - Ex: Ctrl+c para matar processos de foreground (SIGINT)
- Controle de processos
  - Ex: Se um processo de background tenta ler/escrever do terminal, ele pode ser terminado, suspenso. O pai recebe um sinal do kernel
- Notificações
  - Ex: algum dispositivo está pronto
- Alarmes
  - Um processo é notificado de um alarme via sinal (SIGALRM)

Geração de Sinais (2)

Função	kill	2
Cabeçalho	<sys/types.h> <signal.h>	
Protótipo	int kill (pid_t pid , int sig) ;	
Parâmetros	pid : identificação do(s) destinatário(s) (tabela abaixo) sig : sinal	
Resultado	sucesso	fracasso	errno?
	0	-1	sim

pid	processos destinatários
> 0	processo identificado por pid
0	processos pertencentes ao mesmo grupo do emissor
< -1	processos pertencentes ao grupo abs (-pid)

Geração de Sinais (3)

- Exemplo 1: enviar SIGUSR1 ao processo 3423

```
if (kill(3423, SIGUSR1) == -1)
        perror("Failed to send the SIGUSR1 signal");
```

- Exemplo 2: um filho "mata" seu pai

```
if (kill(getppid(), SIGTERM) == -1)
        perror ("Failed to kill parent");
```

- Exemplo 3: enviar um sinal para si próprio

```
if (kill(getpid(), SIGABRT))
        exit(0);
```

Geração de Sinais (4)

- Chamada raise(): um processo envia um sinal para si mesmo

```
#include <signal.h>
int raise(int sig);
```

  - Exemplo 4:

```
if (raise(SIGUSR1) != 0)
                perror("Failed to raise SIGUSR1");
```

- Chamada pause(): Bloqueia execução até a chegada de um sinal

```
#include <unistd.h>
int pause();
```

Tratamento de sinais (1)

- Sinais apresentam um tratamento default
  - abort
    - geração de *core dump*
      - escrita do espaço de endereçamento do processo + conteúdo dos registradores em um arquivo core
    - término do processo
  - exit: término do processo
  - ignore: o sinal é ignorado pelo processo
  - stop: o processo é suspenso
  - continue: "dessuspende" o processo

Tratamento de sinais (2)

- Os usuários podem alterar o tratamento default
  - Definindo seus próprios tratadores (que "capturam" e tratam os sinais)
  - Ignorando alguns sinais
  - Bloqueando sinais temporariamente (o sinal se mantém pendente até que seja desbloqueado)
- Alguns sinais não podem ser bloqueados, ignorados e nem capturados
  - Ex: SIGKILL , SIGSTOP

Tratamento de sinais (3)

- Rotina `signal` permite alterar o comportamento do programa em relação ao recebimento de um sinal específico.

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum,sighandler_t handler);
```

- Exemplo: Como tratar SIGSEGV?
  - Devemos escrever um tratador ...

```
void trata_SIGSEGV(int signum) {
  ...
}
```

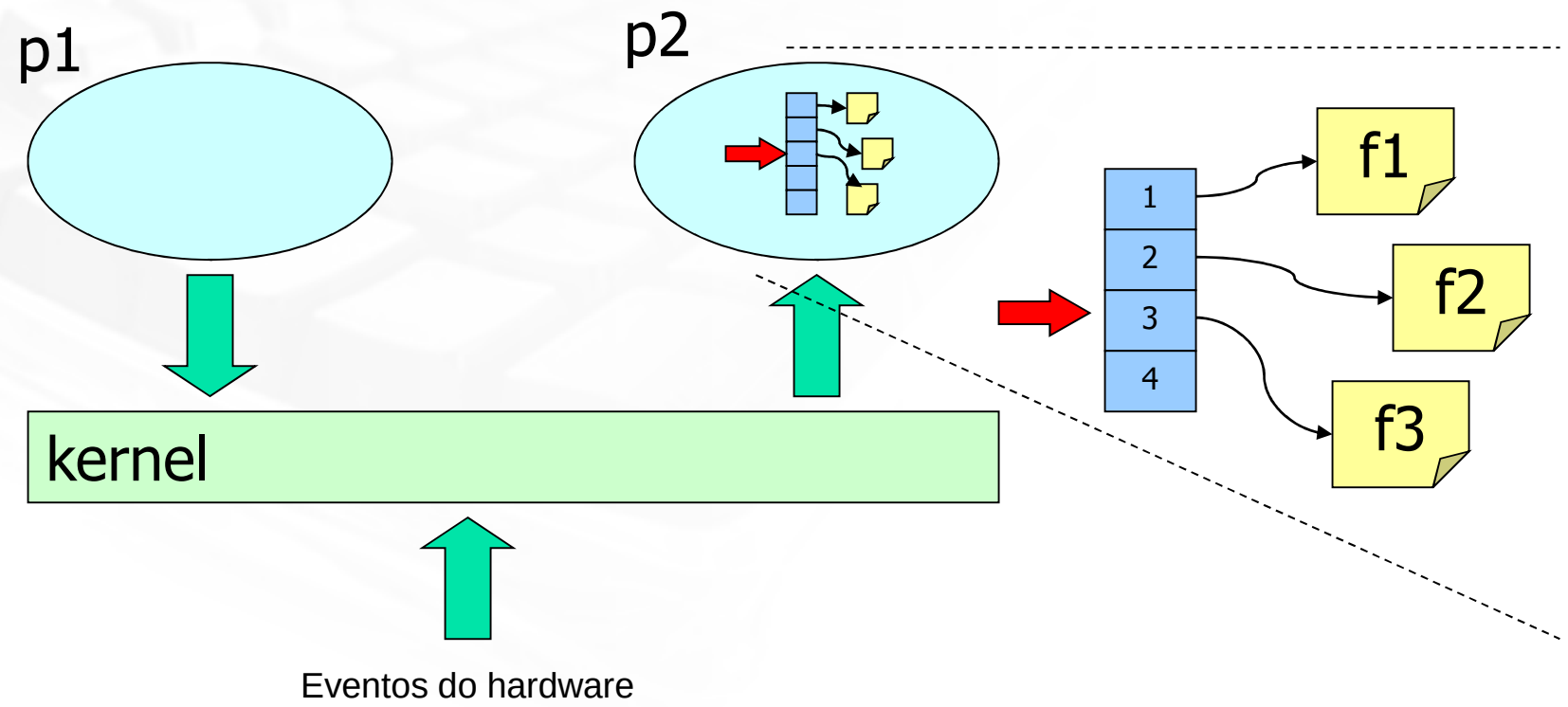
  - ...e instalá-lo

```
signal(SIGSEGV, trata_SIGSEGV);
```

- Como recuperar o tratador default?

```
signal(SIGALRM, SIG_DFL);
```

Tratamento de sinais (4)



- Ativação de funções pré-definidas
  - Vetor de funções registradas

Tratamento de sinais (5)

- Como tratar erros deste tipo?

```
int *px = (int*) 0x01010101;
*px = 0;
```

  - Programa recebe um sinal SIGSEGV
  - O comportamento padrão é terminar o programa
- E erros deste tipo?

```
int i = 3/0;
```

  - Programa recebe um sinal SIGFPE
  - O comportamento padrão é terminar o programa

Tratamento de sinais (6)

```
/* Imprime uma mensagem quando um SIGSEGV é recebido * e restabelece o tratador
padrão. */
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void trata_SIGSEGV(int signum) {
        printf("Acesso indevido `a memória.\n");
        printf("Nao vou esconder este erro. :-)\n");
        signal(SIGSEGV, SIG_DFL);
        raise(SIGSEGV); /* equivale a kill(getpid(), SIGSEGV); */

}

int main() {
        signal(SIGSEGV, trata_SIGSEGV);
        int *px = (int*) 0x01010101;
        *px = 0;
        return 0;
}
```

Tratamento de sinais (6)

- Tratadores pré-definidos
  - `SIG_DFL`: Tratador Default
  - `SIG_IGN`: Ignora o sinal
- Ignorando um sinal
  - Exemplo: para ignorar o Ctrl+c

```
signal(SIGINT, SIG_IGN)
```

Tratamento de sinais (7)

Exemplo de tratadores encadeados

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

void trata_SIGUSR1(int sig) {
  printf("Tratando SIGUSR1.\n");
}

void trata_SIGUSR2 (int sig) {
  printf("Tratando SIGUSR2.\n");
  raise(SIGUSR1);
  printf("Fim do SIGUSR2.\n");
}

int main (void) {

  signal (SIGUSR1, trata_SIGUSR1);
  signal (SIGUSR2, trata_SIGUSR2);

  raise(SIGUSR2);

  sleep(2);
  return 0;
}
```

Tratamento de sinais (6)

- Bloqueando sinais
  - Por que bloquear sinais?
    - Uma aplicação deseja ignorar alguns sinais
    - Evitar condições de corrida quando um sinal ocorre no meio do tratamento de outro sinal
- Como bloquear um sinal?
  - Um conjunto de sinais : `sigprocmask()`
  - Um sinal específico : `sigaction()`

Mais detalhes a seguir...

Manipulando Máscaras e Conjunto de Sinais (1)

- Para representar conjuntos de sinais, usa-se o tipo `sigset_t`
- Rotinas para manipular conjuntos de sinais

```
#include <signal.h>

int sigemptyset(sigset_t *set);
  //Inicializa um conj. vazio

int sigfillset(sigset_t *set);
  //Coloca todos os sinais existentes no conjunto

int sigaddset(sigset_t *set, int signo);
  //adiciona um sinal específico ao conjunto

int sigdelset(sigset_t *set, int signo);
  //remove um sinal específico do conjunto

int sigismember(const sigset_t *set, int signo);
  //verifca se um sinal pertence ao conjunto
```

Manipulando Máscaras e Conjunto de Sinais (2)

- Exemplo: inicializar um conjunto de sinais

```
sigset_t twosigs;
if ((sigemptyset(&twosigs) == -1) ||
    (sigaddset(&twosigs, SIGINT) == -1)  ||
    (sigaddset(&twosigs, SIGQUIT) == -1))
       perror("Failed to set up signal set");
```

Manipulando Máscaras e Conjunto de Sinais (3)

- Todo processo define uma máscara de sinais indicando o conjunto de sinais que estão bloqueados
- Esta máscara é modificada usando-se a rotina

```
#include <signal.h>
int sigprocmask(int how
                const sigset_t *restrict set,
                sigset_t *restrict oset);
```

- O parâmetro `how`:
  - `SIG_BLOCK`: bloqueia os sinais no conj. `set`, adicionando-os à máscara atual.
  - `SIG_UNBLOCK`: desbloqueia os sinais no conj. set, removendo-os da máscara atual
  - `SIG_SETMASK`: substitui a máscara atual.
- Máscara anterior é retornada em `oset`.

Manipulando Máscaras e Conjunto de Sinais (4)

- Exemplo: adcionar SIGINT ao conjunto de sinais bloqueados de um processo

```
sigset_t newsigset;

if ((sigemptyset(&newsigset) == -1) ||
    (sigaddset(&newsigset, SIGINT) == -1))
       perror("Failed to initialize the signal set");
else if (sigprocmask(SIG_BLOCK, &newsigset,NULL) == -1)
       perror("Failed to block SIGINT");
```

Programa que bloqueia e desbloqueia SIGINT

```
int main(int argc,  char *argv[]) {
   int i;
   sigset_t intmask;
   int repeatfactor;
   double y = 0.0;

   if (argc != 2) {
      fprintf(stderr, "Usage: %s repeatfactor\n", argv[0]);
      return 1;
   }
   repeatfactor = atoi(argv[1]);
   if ((sigemptyset(&intmask)==-1) || (sigaddset(&intmask, SIGINT)==-1)){
      perror("Failed to initialize the signal mask");
      return 1;
   }
   for ( ; ; ) {
      if (sigprocmask(SIG_BLOCK, &intmask, NULL) == -1)
         break;
      fprintf(stderr, "SIGINT signal blocked\n");
      for (i = 0; i < repeatfactor; i++)
         y += sin((double)i);
      fprintf(stderr, "Blocked calculation is finished, y = %f\n", y);
      if (sigprocmask(SIG_UNBLOCK, &intmask, NULL) == -1)
         break;
      fprintf(stderr, "SIGINT signal unblocked\n");
      for (i = 0; i < repeatfactor; i++)
         y += sin((double)i);
      fprintf(stderr, "Unblocked calculation is finished, y=%f\n", y);
   }
   perror("Failed to change signal mask");
   return 1;
}
```

... Mais Tratamento de Sinais (1)

- sigaction(): outra forma de definir um tratamento (ou ignorar) de um sinal

```
#include <signal.h>
 int sigaction(int signo,
                const struct sigaction *act,
               struct sigaction *oact);
struct sigaction {
  void (*sa_handler)(int); /* SIG_DFL, SIG_IGN ou ponteiro p/
                              função*/
  sigset_t sa_mask;        /* sinais adicionais a serem
                              bloqueados durante a execução deste
                              tratador */
  int sa_flags;      /* opções especiais, se zerada, sa_handler
                        define a ação a ser executada */

  void(*sa_sigaction) (int, siginfo_t *, void *);
                              /*tratador original*/
};
```

... Mais Tratamento de Sinais (2)

- Exemplo: setar um tratador para o sinal SIGINT

```
void catchctrlc(int signo) {
    char handmsg[] = "I found Ctrl-C\n";
    int msglen = sizeof(handmsg);

    write(STDERR_FILENO, handmsg, msglen);
}
...
struct sigaction act;
act.sa_handler = catchctrlc;
act.sa_flags = 0;

if ((sigemptyset(&act.sa_mask) == -1) ||
     (sigaction(SIGINT, &act, NULL) == -1))
    perror("Failed to set SIGINT to handle Ctrl-C");
```

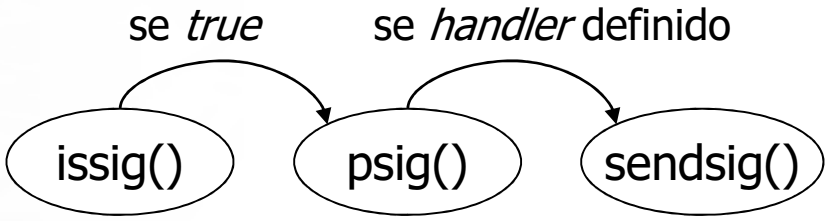
Implementação do Tratamento de Sinais (1)

- O kernel necessita manter algum estado na *U area* e na *proc structure*
- A ***U area*** contém informações necessárias para invocar o manipulador de sinais, incluindo:
  - `u_signal []`: Vetor de tratador de sinais para cada sinal
  - `u_sigmask []`: Máscara de sinais bloqueados para cada tratador
- A ***proc structure*** contém campos associados para geração e transferência de sinais, incluindo:
  - `p_cursig`: o sinal corrente sendo tratado
  - `p_sig`: máscara de sinais pendentes.
  - `p_hold`: máscara de sinais bloqueados
  - `p_ignore`: máscara de sinais ignorados

Implementação do Tratamento de Sinais (2)

- Uma ação de tratamento de um sinal só pode ser executada pelo processo receptor
- Um processo só pode executar o tratamento quando ele estiver *running*
- Se por exemplo um processo possui baixa prioridade, está suspenso ou bloqueado, pode haver um atraso considerável na execução do tratamento do sinal
- O processo receptor fica "a par" de um sinal quando o kernel chama a função *issig()* (em nome do processo)
  - i.e. o processo está no estado kernel running
- Quando issig() é chamada?
  - Imediatamente antes de retornar p/ User Mode (após uma SVC/interrupção/ exceção)
  - Antes de bloquear o processo em algum evento
  - Após acordar um processo (bloqueado em algum evento)

Implementação do Tratamento de Sinais (2)



- Se issig() retornar true, o kernel chama psig() para despachar o sinal
- psig()
  - Termina o processo, gerando core dump se requisitado,

  OU
  - Chama sendsig() para invocar o tratador de sinal definido pelo usuário. sendsig():
    - Retorna o processo para User Mode
    - transfere o controle para o tratador de sinal especificado
    - Faz com que após o tratamento do sinal o processo retome a execução normal (de onde foi interrompido)
- Se o sinal chegou quando o processo estava executando uma SVC, geralmente o kernel aborta a SVC retornando o código de erro EINTR

Referências

- VAHALIA, U. Unix Internals: the new frontiers. Prentice-Hall, 1996.
  - Capítulo 4 (até seção 4.7)
- Deitel H. M.; Deitel P. J.; Choffnes D. R.; "Sistemas Operacionais", 3ª. Edição, Editora Prentice-Hall, 2005
  - Seção 4.7.1
- Silberschatz A. G.; Galvin P. B.; Gagne G.; "Fundamentos de Sistemas Operacionais", 6a. Edição, Editora LTC, 2004.
  - Seção 20.9.1