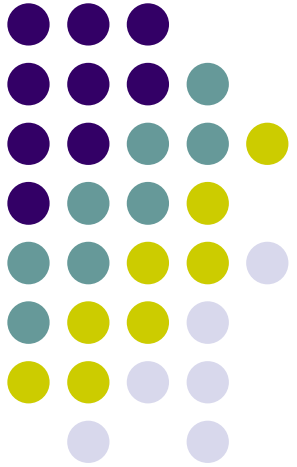
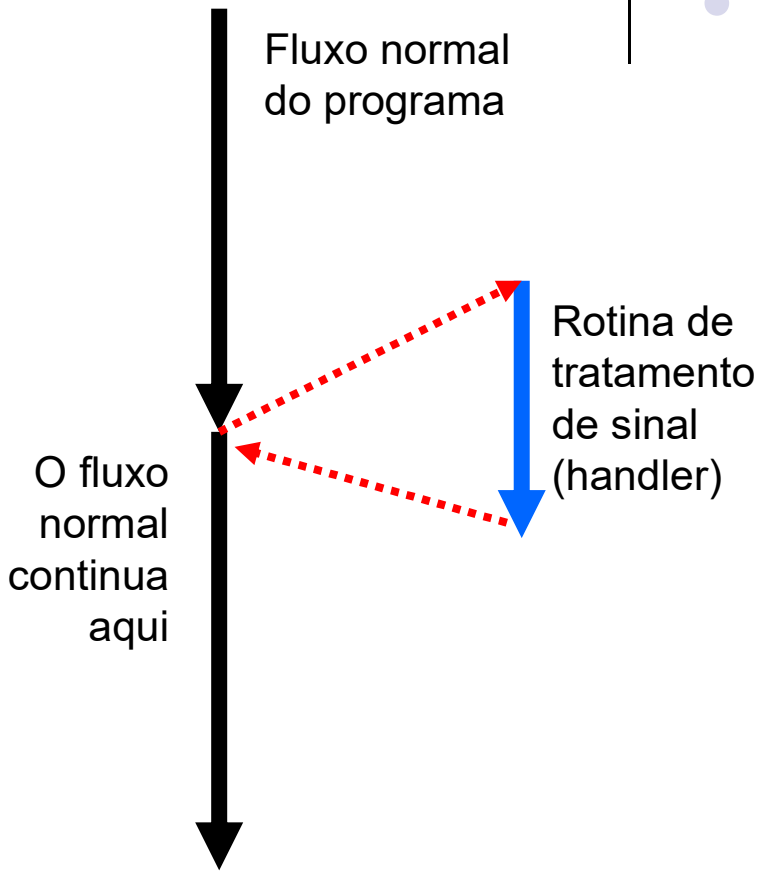
Comunicação entre Processos

Sinais

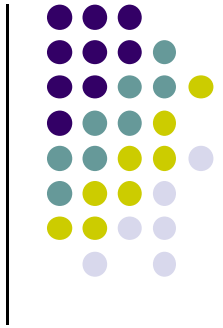


Introdução

- Um sinal é uma notificação assíncrona transmitido entre processos através do Sistema Operacional
- Quando um processo termina anormalmente, ele usualmente tenta enviar um sinal indicando o que deu errado
- Programas C (e o Unix) podem capturar estes sinais para diagnósticos
- Comunicação entre processos podem ser realizadas também através de sinais



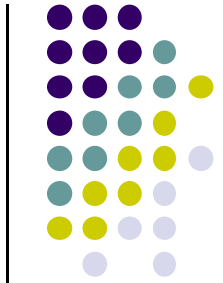
Sinais



- Sinais são interrupções geradas por software que são enviadas para um processo quando acontece um evento

- Cada sinal tem uma ação default que pode ser, entre outras:
  - O sinal é descartado após ser recebido
  - O processo é terminado após o sinal ser recebido

Sinais

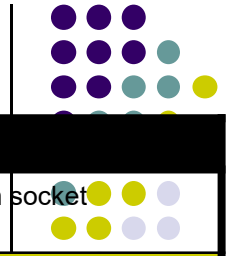


- Cada sinal definido pelo sistema se encaixa em uma das seguintes classes:
  - Hardware
  - Software
  - Notificação de E/S
  - Controle de processos
  - Controle de recursos

Macros são definidas em <signal.h>

- Sinais são numerados de 1 até 31.
- Nenhum sinal tem o número 0
- Todos os sinais tem um nome representando uma constante inteira positiva
  - SIGHUP 1 /* hangup */
  - SIGINT 2 /* interrupt */
  - SIGQUIT 3 /* quit */
  - SIGILL 4 /* instrução ilegal */
  - SIGABRT 6 /* usado por aborto */
  - SIGKILL 9 /* kill de verdade */
  - SIGALRM 14 /* alarme do relógio */
  - SIGCONT 19 /* continuar um processo parado */
  - SIGCHLD 20 /* parar o pai quando o filho parar ou terminar */

Macros são definidas em <signal.h>



Signal	Value	Action	Comment	Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process	SIGURG	16	Ign	Urgent condition on socket (4.2BSD)
SIGINT	2	Term	Interrupt from keyboard	SIGSTOP	17	Stop	Stop process
SIGQUIT	3	Core	Quit from keyboard	SIGTSTP	18	Stop	Stop typed at tty
SIGILL	4	Core	Illegal Instruction	SIGCONT	19	Cont	Continue if stopped
SIGTRAP	5	Core	Trace/breakpoint trap	SIGCHLD	20	Ign	Child stopped or terminated
SIGABRT	6	Core	Abort signal from abort(3)	SIGTTIN	21	Stop	tty input for background process
SIGIOT	6	Core	IOT trap. A synonym for SIGABRT	SIGTTOU	22	Stop	tty output for background process
SIGEMT	7	Term		SIGIO	23	Term	I/O now possible (4.2BSD)
SIGFPE	8	Core	Floating point exception	SIGXCPU	24	Core	CPU time limit exceeded (4.2BSD)
SIGKILL	9	Term	Kill signal	SIGXFSZ	25	Core	File size limit exceeded (4.2BSD)
SIGBUS	10	Core	Bus error (bad memory access)	SIGVTALRM	26	Term	Virtual alarm clock (4.2BSD)
SIGSEGV	11	Core	Invalid memory reference	SIGPROF	27	Term	Profiling timer expired
SIGSYS	12	Core	Bad argument to routine (SVr4)	SIGWINCH	28	Ign	Window resize signal (4.3BSD, Sun)
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers	SIGPWR	29	Term	Power failure (System V)
SIGALRM	14	Term	Timer signal from alarm(2)	SIGUSR1	30	Term	User-defined signal 1
SIGTERM	15	Term	Termination signal	SIGUSR2	31	Term	User-defined signal 2

Term: o processo é terminado
Core: o processo é terminado e um arquivo core é gerado
Ign: o sinal é ignorado
Stop: o processo para (não termina – vai dormir)
Cont: continuar depois de um Stop

Erro
Aplicativos
Controle
Alarme
outros

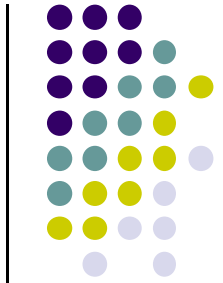
Enviando sinais

- Existem duas formas de enviar sinais a partir de um processo:
  - int kill(pid_t pid, int sig)

    Uma system call que envia um sinal para o processo pid
    - Se pid > 0, o sinal é enviado ao processo pid
    - Se pid = 0, o sinal é enviado a todos os processos do grupo ao qual pertence o processo corrente
    - Se pid = -1, o sinal é enviado a todos os processos para os quais o processo atual tem permissão de fazer (exceto para o processo 1)
    - Se pid < -1, o sinal é enviado a todos os processos do grupo -pid
  - int raise(int sig)

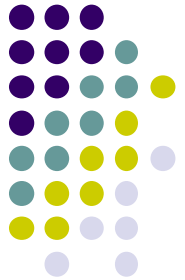
    Envia um sinal para o processo que está sendo executado
  - Definidos em <signal.h>
  - Retornam:
    - Em caso de sucesso: 0 (zero)
    - Em caso de falha: kill retorna -1

Enviando sinais



- Existe também um comando Unix chamado kill que pode ser utilizado para enviar sinais a partir da linha de comando
  - $ kill –s sinal pid

Enviando sinais

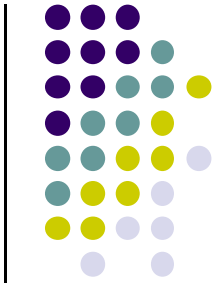


- Um sinal ocorre em momentos aparentemente aleatórios para o processo
- Um processo não pode simplesmente testar o valor de uma variável para verificar se um sinal ocorreu
- O processo deve informar ao kernel o que fazer quando um sinal ocorrer

Enviando sinais

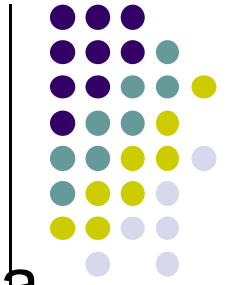
- Sinais podem ser enviados a um processo através do teclado
  - Ctrl-C
    - Envia o sinal SIGINT
    - Causa o término imediato do processo
  - Ctrl-Z
    - Envia o sinal SIGTSTP
    - Causa a suspensão do processo
  - Ctrl-\
    - Envia o sinal SIGABRT
    - Causa o término imediato do processo
    - Ctrl-\ possui o mesmo efeito que Ctrl-C através de outro sinal
- Somente afetam processos em foreground

Manipulação de sinais



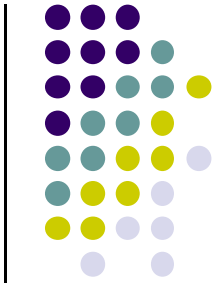
- Um processo pode tratar sinais das seguintes formas:
  - O processo pode deixar que a ação default aconteça (SIG_DFL)
  - O processo pode bloquear o sinal (SIG_IGN) (SIGKILL não pode ser ignorado)
  - O processo pode capturar o sinal através de um handler (uma função manipuladora)
- Um handler é uma função que um processo especificou para manipular determinado sinal

Manipulação de sinais



- Uma função handler é executada através de uma chamada implícita de função (quando um sinal é recebido)
- A função retorna para o ponto de execução onde ocorreu a interrupção
- O handler executa no mesmo contexto de memória do processo normal
- Se houver necessidade de troca de contexto para o tratamento do sinal, o processo deve realizar a troca por conta própria

Função signal( )

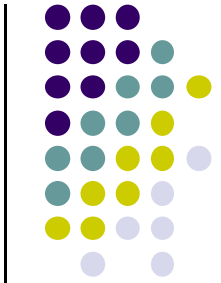


- Configura o tratamento de sinais
  - Definida em <signal.h>

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

  - Onde:
    - signum: número do sinal a ser tratado
    - func: endereço da função de tratamento do sinal ou SIG_IGN (ignorar o sinal) ou SIG_DFL (tratamento default)
  - Retorna
    - Em caso de sucesso, o endereço da antiga função de tratamento
    - Em caso de falha, -1

Configurando manipuladores



- Exemplos:
  - Para ignorar um Ctrl-C:

    signal(SIGINT, SIG_IGN);

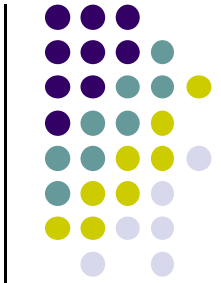
  - Para permitir que o processo volte a aceitar Ctrl-C para terminar o processo:

    signal(SIGINT, SIG_DFL);

  - Para desviar para uma função quando Ctrl-C for digitado:

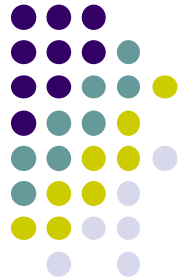
    signal(SIGINT, trataCtrlC);

Exemplo: configurar rotina de atendimento



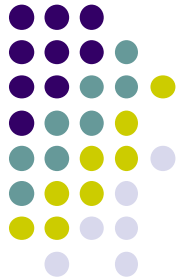
```
#include <...>

int main (void)
{
   // captura Ctrl-C
   if (signal(SIGINT, rotinaDeAtendimento) == SIG_ERR)
   {
      // erro ao instalar a rotina de atendimento do sinal SIGINT
                ...
   }
   ...
}

void rotinaDeAtendimento(int sinal)
{
   // tratamento do(s) sinal(is)
   ...
}
```

Exemplo: ignorar sinal



```
#include <...>

int main (void)
{
   // simplesmente ignora Ctrl-C
   if (signal(SIGINT, SIG_IGN) == SIG_ERR)
   {
      // erro ao instalar a rotina de atendimento do sinal SIGINT
                ...
   }
   ...
}
```

Exemplo: mesma rotina para tratar vários sinais



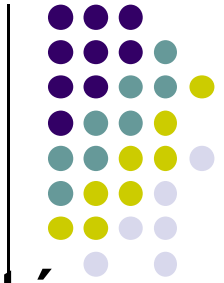
```
#include <...>

int main (void)
{
   signal(SIGINT, rotinaDeAtendimento);
   signal(SIGUSR1, rotinaDeAtendimento);
   ...
}

void rotinaDeAtendimento(int sinal)
{
   // tratamento do(s) sinal(is)
   switch(sinal)
   {
                case SIGINT:
                        ...
                        break;
                case SIGUSR1:
                        ...
                        break;
   }
}
```

ctrl-C.c

```
sinais$ ./ctrl-c
Endereco do manipulador anterior (nil)
Endereco do manipulador anterior (nil)
Ctrl-C desabilitado. Use Ctrl-\ para terminar
Você precionou Ctrl-C
Você precionou Ctrl-C
Terminando o processo...
sinais$
```

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

#define EVER ;;

void intHandler(int sinal);
void quitHandler(int sinal);

int main (void)
{
   void (*p)(int);// ponteiro para função que recebe int como parâmetro
   p = signal(SIGINT, intHandler);
   printf("Endereco do manipulador anterior %p\n", p);
   p = signal(SIGQUIT, quitHandler);
   printf("Endereco do manipulador anterior %p\n", p);
   puts ("Ctrl-C desabilitado. Use Ctrl-\\ para terminar");
   for(EVER);
}
void intHandler(int sinal)
{
   puts ("Você pressionou Ctrl-C (%d)", sinal);
}
void quitHandler(int sinal)
{
   puts ("Terminando o processo...");
   exit (0);
}
```

Parar um processo

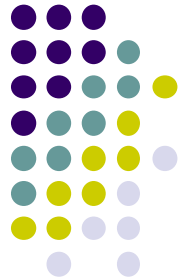


- Coloca o processo corrente para dormir até que este receba um sinal que o termine ou cause a sua continuação
  - Definido em <unistd.h>

  int pause(void);

  - Retorna:
    - Em caso de erro, -1
    - Em caso de sucesso, não retorna ☺

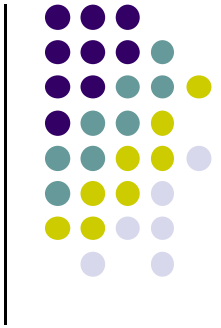
Exemplo de pause( )



```c
#include <stdio.h>
#include <unistd.h>
int main (void)
{
   puts ("vou parar...");
   pause();
   puts ("Continuei!");
   return 0;
}
```

```
sinais$ ./pausa &
vou parar...
[1] 6448
sinais$ ps
  PID TTY          TIME CMD
 6184 pts/0    00:00:00 bash
 6431 pts/0    00:00:00 bash
 6448 pts/0    00:00:00 pausa
 6449 pts/0    00:00:00 ps
sinais$ kill -s SIGUSR1 6448
sinais$ ps
  PID TTY          TIME CMD
 6184 pts/0    00:00:00 bash
 6431 pts/0    00:00:00 bash
 6450 pts/0    00:00:00 ps
[1]+  User defined signal 1   ./pausa
sinais$
```

Exemplo de pause( )



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

void handler(int sinal);

int main (void)
{
   signal(SIGUSR1, handler);

   puts ("vou parar...");
   pause();
   puts ("Continuei!");

   return 0;
}

void handler(int sinal)
{
   printf ("Sinal %d recebido\n", sinal);
}
```

```
sinais$ ./pause2 &
vou parar...
[1] 6554
sinais$ ps
  PID TTY          TIME CMD
 6184 pts/0    00:00:00 bash
 6431 pts/0    00:00:00 bash
 6554 pts/0    00:00:00 pause2
 6555 pts/0    00:00:00 ps
sinais$ kill -s SIGUSR1 6554
Sinal 10 recebido
Continuei!
sinais$
```

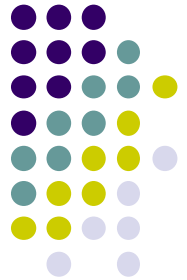
Uso de alarme

- Programa a entrega de um sinal SIGALARM ao processo ao final de n segundos
- Definida em <unistd.h>

unsigned alarm(unsigned alarm);

- Onde:
  - alarm: tempo em segundos para disparar o alarme
  - Se alarm = 0, desarma o alarme
- Retorna
  - O tempo em segundos até a ativação do alarme

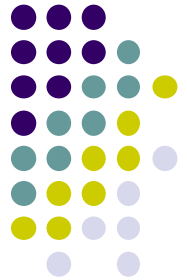
Exemplo de alarme



```
sinais$ ./alarme &
[1] 6636
sinais$ 10 segundos
10 segundos
10 segundos
```

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

#define TEMPO 10
#define EVER ;;

void trataAlarme(int sinal);

int main (void)
{
   signal (SIGALRM, trataAlarme);
   alarm (TEMPO);
   for(EVER);
   return 0;
}

void trataAlarme(int sinal)
{
   printf ("%d segundos\n", TEMPO);
   alarm (TEMPO);
}
```

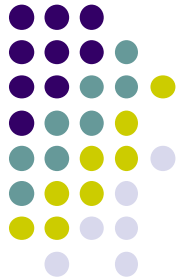
Função sleep( )



- Coloca o processo corrente para dormir até passar o tempo especificado ou até o processo receber um sinal que não seja ignorado
  - Definida em <unistd.h>

unsigned sleep(int segundos);

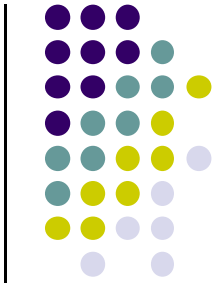
Função abort( )



- Gera uma terminação anormal do programa
- Envia um sinal SIGABRT
- O sinal pode ser tratado por um handler
- Definido em <stdlib.h>

```
void abort(void);
```

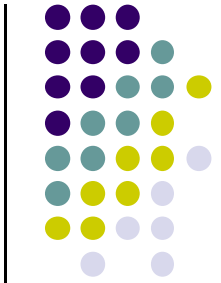
Tratamento de sinais após fork()/exec



- Após fork()
  - O tratamento dos sinais é herdado pelo filho
  - O filho pode alterar o tratamento do sinal
- Após exec()
  - Se o sinal estiver programado para ser ignorado
    - O processo continua a ignorar o sinal
  - Se o sinal não estiver programado para ser ignorado
    - O sinal terá o tratamento padrão

Perguntas?

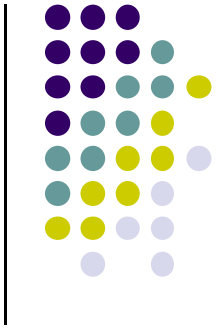
Lab. sobre Sinais



Outro Exemplo do uso de Sinais...

- Programa que executa um fork() gerando um filho e espera determinado tempo para que o programa-filho termine. Se o programa filho não terminar, o programa pai termina a execução do programa filho
- Exemplo:

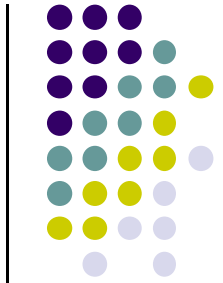
  $ filhocidio tempo programafilho arg1 arg2 ... argn



filhocidio.c

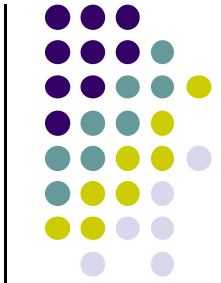
```c
#include <...>
void childhandler(int signo);
int delay;
int main (int argc, char *argv[])
{
    pid_t pid;

    signal(SIGCHLD, childhandler);
    if ((pid = fork()) < 0)
    {
                fprintf(stderr, "Erro ao criar filho\n");
                exit(-1);
    }
    if (pid == 0) /* child */
                for(EVER); /* ou sleep(3);*/
    else /* parent */
    {
                sscanf(argv[1], "%d", &delay); /* read delay from command line */
                sleep(delay);
                printf("Program %s exceeded limit of %d seconds!\n", argv[2], delay);
                kill(pid, SIGKILL);
    }
    return 0;
}
void childhandler(int signo) /* Executed if child dies before parent */
{
    int status;
    pid_t pid = wait(&status);
    printf("Child %d terminated within %d seconds com estado %d.\n", pid, delay, status);
    exit(0);
}
```

Lab. sobre sinais



1) Explique o funcionamento do programa filhocídio.c.
2) Altere o programa filhocidio.c para que o filho, em vez de executar o for(EVER) dê um exec() no programa sleep5 e depois no programa sleep15 indicados a seguir. Execute cada um e explique o funcionamento desta nova versão do programa filhocidio.c

Execução de filhocidio.c

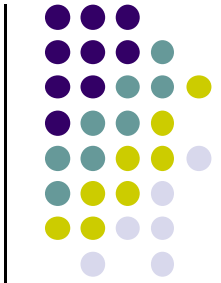


```c
#include <stdio.h>
#include <unistd.h>
int main (void)
{
   fprintf(stdout, "indo dormir...\n");
   sleep(5);
   fprintf(stdout, "Acordei!\n");
   return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>
int main (void)
{
   fprintf(stdout, "indo dormir...\n");
   sleep(15);
   fprintf(stdout, "Acordei!\n");
   return 0;
}
```

```
sinais$ ./filhocidio 10 ./sleep5
indo dormir...
Acordei!
Child 6580 terminated within 10 seconds com estado 0.
sinais$ ./filhocidio 10 ./sleep15
indo dormir...
Program ./sleep15 exceeded limit of 10 seconds!
sinais$
```

Lab. sobre sinais



3) Faça um programa para monitorar e informar o preço de chamadas telefônicas. O programa deverá ser executado em background.

- O início e o término de uma chamada são informados através dos sinais USR1 e USR2, respectivamente.
- O custo da ligação é de 2 centavos por segundo, para ligações de até 1 minuto ou de 1 centavo por segundo a partir do 2º. minuto, ou seja, uma ligação de 1m30s custa R$1,50.