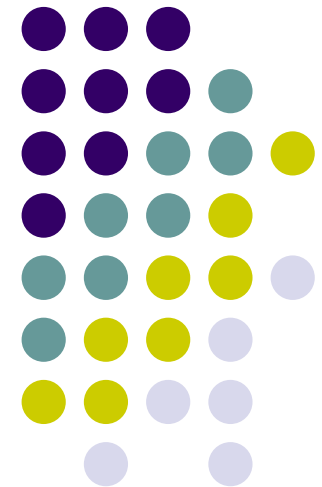


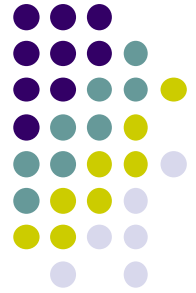
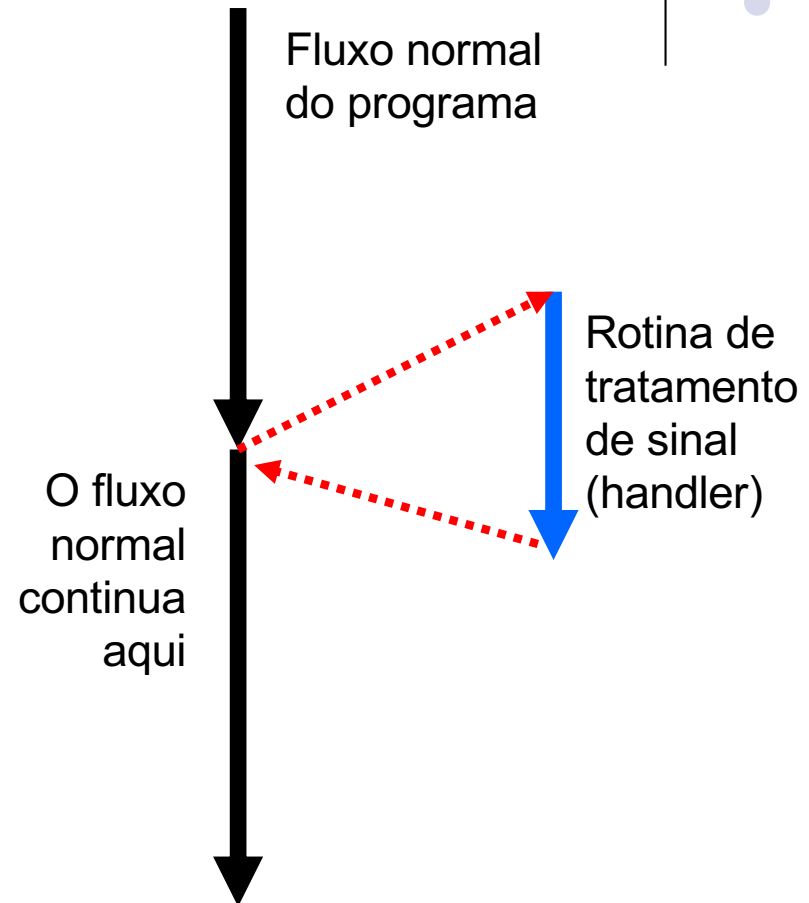
# Comunicação entre Processos

Sinais

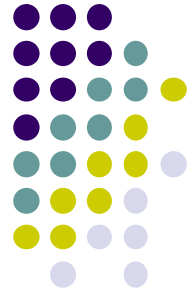


# Introdução

- Um sinal é uma notificação assíncrona transmitida entre processos através do Sistema Operacional
- Quando um processo termina, ele usualmente gera um sinal indicando o que deu errado
- Programas C (e o Unix) podem captar estes sinais para diagnósticos
- Sinais tb podem ser usados como forma (primitiva) de comunicação entre processos



# Sinais



- Sinais são interrupções geradas por software (o núcleo, em nome de um processo) 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

# Macros são definidas em `<signal.h>`



- Sinais são numerados de 1 até 31 (não há sinal de 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 \*/
- **Atenção: Os nomes podem diferir em versões do UNIX.**
- Use `$ kill -l` para ver os nomes.

# SIGABRT (signal 6)



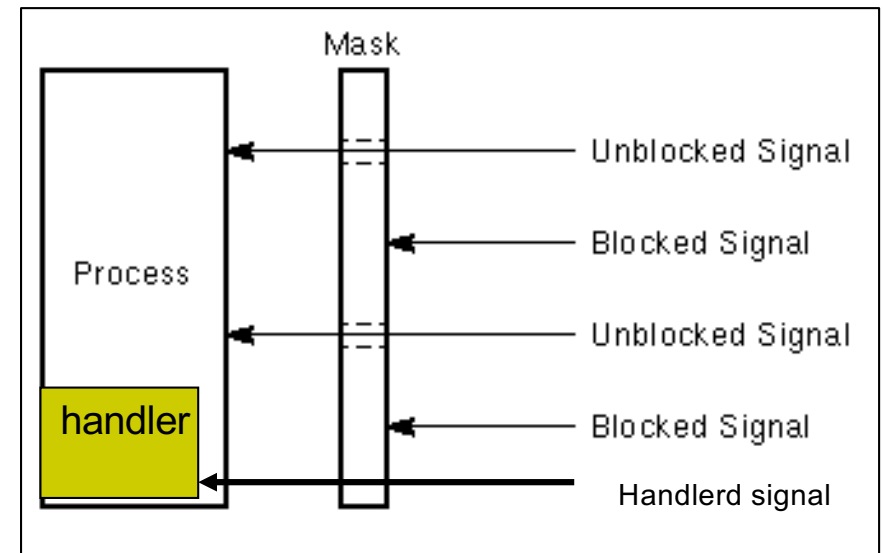
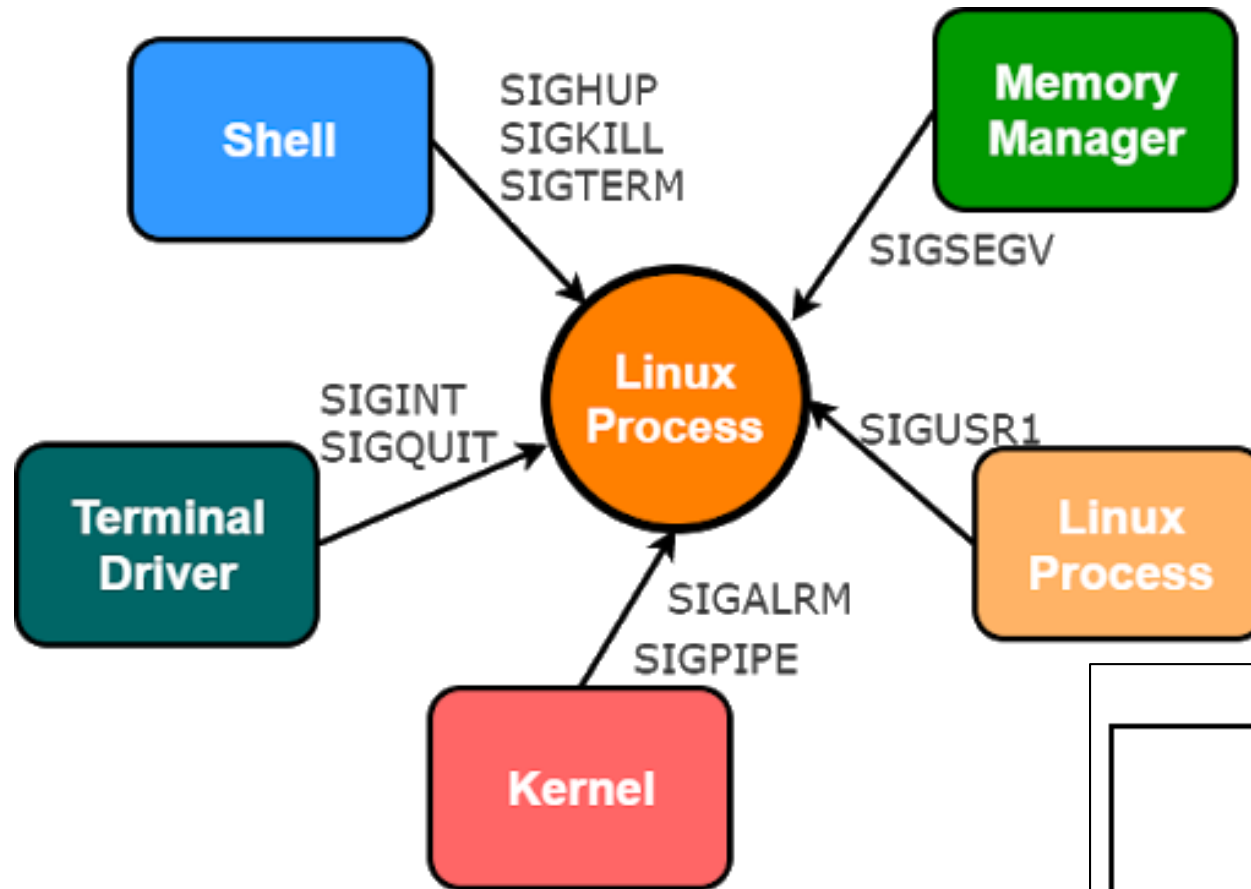
É gerado por um processo ao detectar um erro interno ou alguma restrição (*constraint*) violada.

.

Por exemplo:

- `malloc()` em `libc.so` chamará `abort()` se suas estruturas internas forem danificadas por um estouro de heap.
- Ou ao encontrar um erro limpando um cache

# Origem de alguns sinais



# 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 pára (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 próprio processo
- 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
- Exemplo:
  - \$ kill -s SIGUSR1 985      /\* envio para pid 985 \*/

# 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-\ul>    - 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

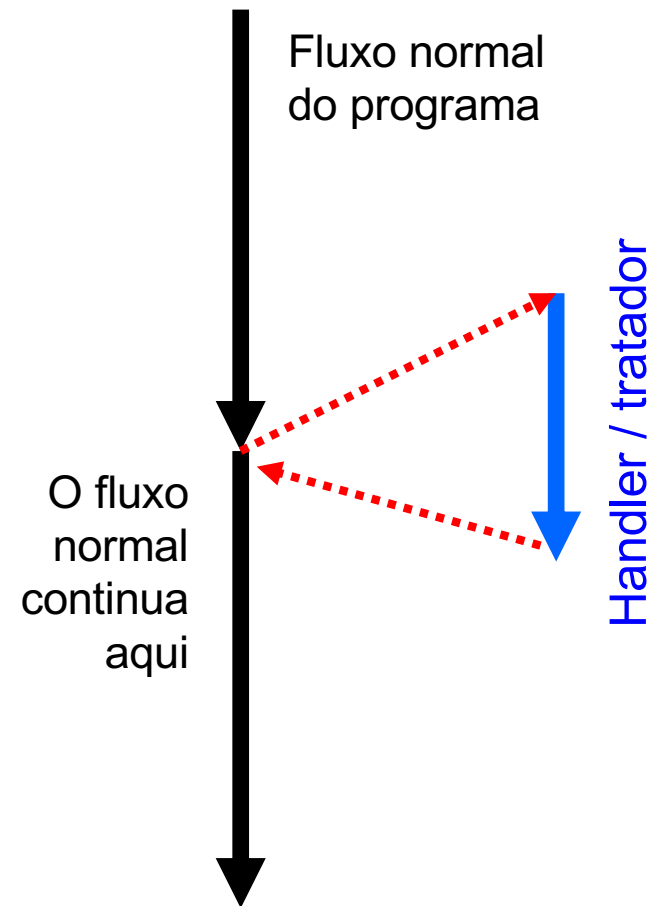


- Um processo pode tratar sinais das seguintes formas:
  - O processo pode deixar que a **ação default** aconteça (SIG\_DFL)
  - O processo **pode bloquear o sinal** (SIG\_IGN) (SIGKILL não pode ser ignorado)
  - O processo **pode capturar o sinal** através de um handler (um tratador)
- Um tratador (handler) é um procedimento que o programador criou para processar 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



# Função signal( )



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

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

- Onde:
  - signum: número/nome 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);    /* ignorando sinal */
```

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

```
signal(SIGINT, SIG_DFL);    /* tratamento default */
```

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

```
signal(SIGINT, trataCtrlC); /* tratando sinal */
```

# Exemplo: configurar função de tratamento



```
#include <...>
```

```
int main (void)
{
```

```
    // captura Ctrl-C
```

```
    if (signal(SIGINT, funcaoTratadora) == SIG_ERR)
```

```
    {
```

```
        // erro ao instalar a rotina de atendimento do sinal SIGINT
```

```
        ...
```

```
    }
```

```
    ...
```

```
}
```

```
void funcaoTratadora (int sinal)
```

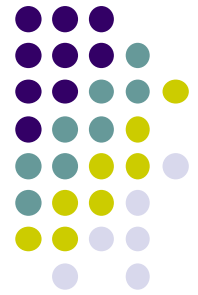
```
{
```

```
    // tratamento do(s) sinal(is)
```

```
    ...
```

```
}
```

# Exemplo: mesma rotina para tratar vários sinais



```
#include <...>

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

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



```
#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)
```

```
{
```

```
    printf("Você pressionou Ctrl-C (%d) \n", sinal);
```

```
}
```

```
void quitHandler(int sinal)
```

```
{
```

```
    printf("Terminando o processo...\n");
```

```
    exit (0);
```

```
}
```

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



ctrl-c.c

# 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 😊

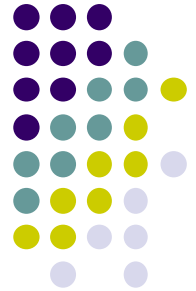
# Exemplo de pause( )



```
#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( )



```
#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

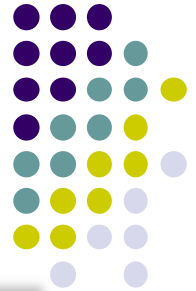
- Agenda a entrega de um sinal SIGALARM ao processo ao final de n segundos
- Definida em <unistd.h>

```
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



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

#define TEMPO 10
#define EVER ;;

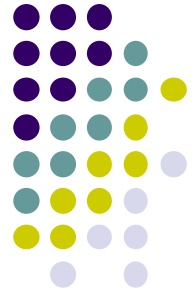
void trataAlarme(int sinal);

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

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

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

# 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);`



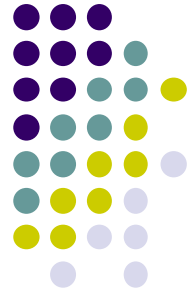
# Tratamento de sinais após `fork()/exec`



- 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



# Exemplo



- Programa que executa um outro e espera determinado tempo para que o programa-filho termine. Se o programa filho não terminar, o pai mata o filho
- Exemplo:  
\$ filhomicidio tempo programafilho arg1 arg2 ... argn

```

#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 */
        execve(argv[2], 0, 0); /* 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);
        sleep(1); /* necessary for SIGCHLD to arrive */
    }
    return 0;
}

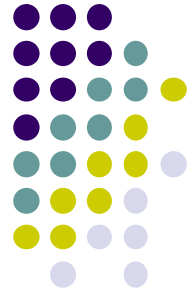
void childhandler(int signo) /* Executed if child dies before parent */
{
    int status;
    pid_t pid = wait(&status);
    printf("Child %d terminated within %d seconds com estado %d.\n", pid, delay, status);
    exit(0);
}

```



filhomicidio.c

# Execução de filhocidio.c



```
#include <stdio.h>
#include <unistd.h>
int main (void)
```

```
{
    fprintf(stdout, "indo dormir...\n");
    sleep(5);
    fprintf(stdout, "Acordei!\n");
    return 0;
}
```

```
-----
#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$ █
```

# Perguntas?



# Exercícios sobre sinais



1) Execute o programa “ctrl-c.c”.

Digite Ctrl-C e Ctrl-\. Analise o resultado.

Neste mesmo programa, remova os comandos `signal( )` e repita o teste anterior observando os resultados.

# Exercícios sobre sinais



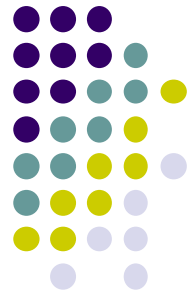
- 2) Tente fazer um programa para interceptar o sinal SIGKILL. Você conseguiu? Explique.
  - 3) Execute e explique o funcionamento de `filhocidio.c`
  - 4) Usando SIGSTOP e SIGCONT faça um programa que crie 2 processos filho e alterne a execução dos filhos. Após 10 trocas de contexto, o processo pai mata os processos filho.
- OBS: Os processos filho são formados por loops infinitos.

# Exercícios sobre sinais – Lab 4



- 5) Faça um programa que leia 2 números e imprima o resultado das 4 operações básicas sobre estes 2 números.
- Verifique o que acontece se o 2º. número da entrada for 0 (zero)
  - Capture o sinal de erro de floating point (SIGFPE) e repita a experiência anterior

# Exercícios sobre sinais – Lab 4



- 6) 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 SIGUSR1 e SIGUSR2, 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.



# Exercício sobre sinais – Lab 4



7) Elabore três programas I/O bound que não terminem (loop de msgs no vídeo).

- Elabore um programa que seja capaz de executar os 3 programas indicados anteriormente e que simule o compartilhamento da CPU entre os 3 processos com escalonamento Round-Robin com uma fatia de tempo de 1 segundo para o primeiro processo e de 2 segundos para os demais processos. Execute os programas e relate o que aconteceu