Interacção por sinais no Unix

José C. Cunha , DI-FCT-UNL

Referência:

Capítulo 6, UNIX System Programming,
K. Haviland, D. Gray, B. Salama,
Addison-Wesley, 1999

Notificações assíncronas (sinais)

- **Informar de situações imprevisíveis:**
  - **podem ou não ocorrer durante a execução**
  - **se ocorrerem, não se sabe quando**

Notificações assíncronas (sinais)

- **Informar de situações imprevisíveis:**
  - **podem ou não ocorrer durante a execução**
  - **se ocorrerem, não se sabe quando**
- **Exemplos:**
  - **erros inesperados no programa (violação da protecção de memória, divisão por zero…)**

Notificações assíncronas (sinais)

- **Informar de situações imprevisíveis:**
  - **podem ou não ocorrer durante a execução**
  - **se ocorrerem, não se sabe quando**
- **Exemplos:**
  - **erros inesperados no programa (violação da protecção de memória, divisão por zero…)**
  - **acções desencadeadas pelo utilizador (ex: pede para abortar o processo)**

Notificações assíncronas (sinais)

- **Informar de situações imprevisíveis:**
  - **podem ou não ocorrer durante a execução**
  - **se ocorrerem, não se sabe quando**
- **Exemplos:**
  - **erros inesperados no programa (violação da protecção de memória, divisão por zero…)**
  - **acções desencadeadas pelo utilizador (ex: pede para abortar o processo)**
  - **um processo notifica outro sobre qualquer evento**

Sinais no sistema Unix

- **Mecanismo *software* semelhante às interrupções do *hardware***
  - **oferecido pelo SO aos processos**
  - **permitindo a estes reagirem à ocorrência de um evento**

Sinais no sistema Unix

- **Mecanismo *software* semelhante às interrupções do *hardware***
  - **oferecido pelo SO aos processos**
  - **permitindo a estes reagirem à ocorrência de um evento**
- **Inclui apenas um identificador (tipo do sinal)**

Sinais no sistema Unix

- **Mecanismo *software* semelhante às interrupções do *hardware***
  - **oferecido pelo SO aos processos**
  - **permitindo a estes reagirem à ocorrência de um evento**
- **Inclui apenas um identificador (tipo do sinal)**
- **O envio dum sinal é assíncrono**
  - **podem ser gerados/enviados independentemente da execução do programa do processo "receptor"**

Sinais no sistema Unix

- **Mecanismo *software* semelhante às interrupções do *hardware***
  - **oferecido pelo SO aos processos**
  - **permitindo a estes reagirem à ocorrência de um evento**
- **Inclui apenas um identificador (tipo do sinal)**
- **O envio dum sinal é assíncrono**
  - **podem ser gerados/enviados independentemente da execução do programa do processo "receptor"**
- **e a recepção é implícita**
  - **uma função no receptor (*handler*) é executada quando o sinal ocorre**

Geração de sinais

- **Um sinal é gerado pelo SO como consequência de um evento:**

Geração de sinais

- **Um sinal é gerado pelo SO como consequência de um evento:**
  - **provocado pelo próprio processo:**
    - `alarm(int sec)`, `abort( )`

Geração de sinais

- **Um sinal é gerado pelo SO como consequência de um evento:**
  - **provocado pelo próprio processo:**
    - `alarm(int sec)`, `abort( )`
  - **acontecimentos detectados pelos dispositivos *hardware*:**
    - **FPU- divisão por zero; MMU- endereço inválido**

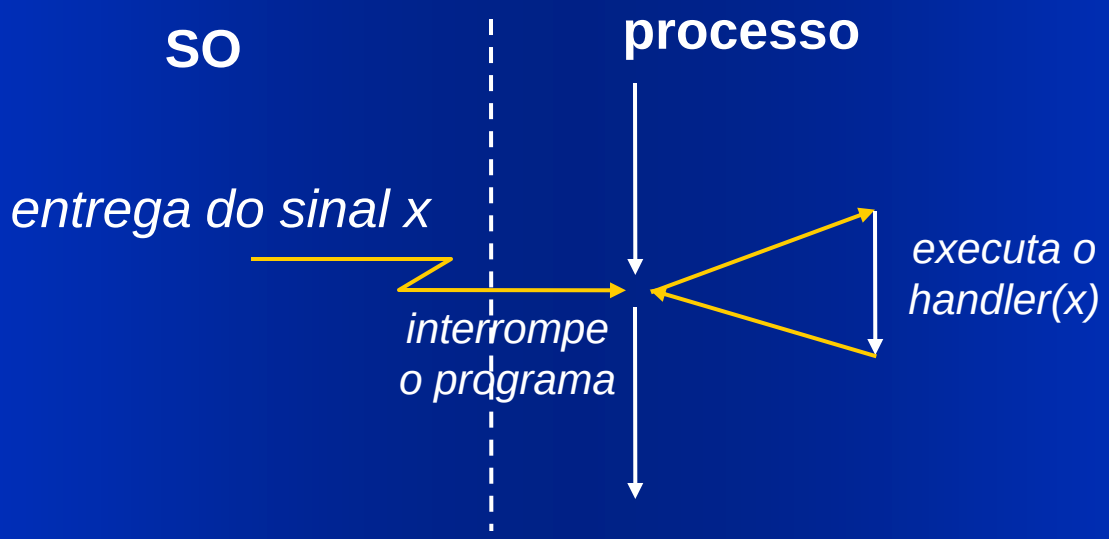
Geração de sinais

- **Um sinal é gerado pelo SO como consequência de um evento:**
  - **provocado pelo próprio processo:**
    - `alarm(int sec)`, `abort( )`
  - **acontecimentos detectados pelos dispositivos *hardware*:**
    - **FPU- divisão por zero; MMU- endereço inválido**
  - **acontecimentos detectados pelo SO:**
    - **terminação de um processo**

Geração de sinais

- **Um sinal é gerado pelo SO como consequência de um evento:**
  - **provocado pelo próprio processo:**
    - ***alarm(int sec)*, *abort( )***
  - **acontecimentos detectados pelos dispositivos *hardware*:**
    - **FPU- divisão por zero; MMU- endereço inválido**
  - **acontecimentos detectados pelo SO:**
    - **terminação de um processo**
  - **chamada ao sistema por um processo:**
    - ***kill(int pid, int sig)* --- ENVIAR sinal**

Tratamento de sinais (recepção)



Handler **- definido pela chamada ao SO (senão tem uma definição pré-definida pelo SO para cada tipo de sinal):**

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

Tratamento de sinais (recepção)

- **A ideia do *handler* é permitir alguma reacção no destino:**

Tratamento de sinais (recepção)

- **A ideia do *handler* é permitir alguma reacção no destino:**
  - **p.ex. alterar *flags*; mudar contadores; etc.**

Tratamento de sinais (recepção)

- **A ideia do *handler* é permitir alguma reacção no destino:**
  - **p.ex. alterar *flags*; mudar contadores; etc.**
  - **se é necessária mais informação , o programa deve depois usar um mecanismo de comunicação, para a obter**

Tratamento de sinais (recepção)

- **A ideia do *handler* é permitir alguma reacção no destino:**
  - **p.ex. alterar *flags*; mudar contadores; etc.**
  - **se é necessária mais informação , o programa deve depois usar um mecanismo de comunicação, para a obter**
- ***"Handlers"* especiais:**

Tratamento de sinais (recepção)

- **A ideia do *handler* é permitir alguma reacção no destino:**
  - **p.ex. alterar *flags*; mudar contadores; etc.**
  - **se é necessária mais informação , o programa deve depois usar um mecanismo de comunicação, para a obter**
- ***"Handlers"* especiais:**
  - **SIG_IGN(ore): ignorar sinal (nem todos os sinais podem ser ignorados)**

Tratamento de sinais (recepção)

- **A ideia do *handler* é permitir alguma reacção no destino:**
  - **p.ex. alterar *flags*; mudar contadores; etc.**
  - **se é necessária mais informação , o programa deve depois usar um mecanismo de comunicação, para a obter**
- ***"Handlers"* especiais:**
  - **SIG_IGN(ore): ignorar sinal (nem todos os sinais podem ser ignorados)**
  - **SIG_D(e)F(au)L(t) : repor o tratamento *implícito (default)* pré-definido pelo SO**

Tratamento *default* (pré-definido)

- **Para a maior parte dos sinais, o tratamento pré-definido (*default)* no SO, é terminar anormalmente o processo**

Tratamento *default* (pré-definido)

- **Para a maior parte dos sinais, o tratamento pré-definido (*default)* no SO, é terminar anormalmente o processo**
  - **ex: os sinais para situações anómalas: (ver o manual)**
    - **SIGFPE – erro detectado pela FPU**

Tratamento *default* (pré-definido)

- **Para a maior parte dos sinais, o tratamento pré-definido (*default)* no SO, é terminar anormalmente o processo**
  - **ex: os sinais para situações anómalas: (ver o manual)**
    - **SIGFPE – erro detectado pela FPU**
    - **SIGSEGV – erro detectado pela MMU**

Tratamento *default* (pré-definido)

- **Para a maior parte dos sinais, o tratamento pré-definido (*default)* no SO, é terminar anormalmente o processo**
  - **ex: os sinais para situações anómalas: (ver o manual)**
    - **SIGFPE – erro detectado pela FPU**
    - **SIGSEGV – erro detectado pela MMU**
    - **SIGILL – erro detectado pelo CPU**

Tratamento *default* (pré-definido)

- **Para a maior parte dos sinais, o tratamento pré-definido (*default)* no SO, é terminar anormalmente o processo**
  - **ex: os sinais para situações anómalas: (ver o manual)**
    - **SIGFPE – erro detectado pela FPU**
    - **SIGSEGV – erro detectado pela MMU**
    - **SIGILL – erro detectado pelo CPU**
    - **SIGINT – pedido de interrupção**
      (p.ex. ^C no terminal)

Tratamento *default* (pré-definido)

- **Para a maior parte dos sinais, o tratamento pré-definido (*default)* no SO, é terminar anormalmente o processo**
  - **ex: os sinais para situações anómalas: (ver o manual)**
    - **SIGFPE – erro detectado pela FPU**
    - **SIGSEGV – erro detectado pela MMU**
    - **SIGILL – erro detectado pelo CPU**
    - **SIGINT – pedido de interrupção**
      (p.ex. ^C no terminal)
    - **SIGKILL – terminação forçada (não pode ser mudado!)**

Tratamento *default* (pré-definido)

- **Para a maior parte dos sinais, o tratamento pré-definido (*default)* no SO, é terminar anormalmente o processo**
  - **ex: os sinais para situações anómalas: (ver o manual)**
    - **SIGFPE – erro detectado pela FPU**
    - **SIGSEGV – erro detectado pela MMU**
    - **SIGILL – erro detectado pelo CPU**
    - **SIGINT – pedido de interrupção**
      (p.ex. ^C no terminal)
    - **SIGKILL – terminação forçada (não pode ser mudado!)**
- **Para outros o tratamento pré-definido é ignorar**
  - **ex: SIGCHLD – um processo filho terminou**

Terminação do processo por um sinal

- **É como se invocasse *_exit()***
- **O SO passa um indicador do estado de terminação do filho para o pai**
  - **no pai este pode ser obtido assim:**

```
wait( &status );
if ( WIFSIGNALED(status) )
  printf("o filho terminou com o sinal %d\n",
                    WTERMSIG(status) );
```

- **Se a terminação por um sinal for de um erro, é gerado um *core dump*: ficheiro com a imagem binária do mapa de memória do processo**

Exemplo: tratar SIGCHLD

- **Quando um processo termina, o seu pai recebe SIGCHLD. Tratamento no pai:**

```
void tratafilho(int sig)
{ wait(NULL); }
```

```
int main(…)
{
signal(SIGCHLD, tratafilho);
…
```

(não funciona da mesma maneira em todas as versões UNIX)

Exemplo: tratar o SIGINT

- **Pode ser gerado pelo terminal, p.ex. com tecla CONTROL-C**

```
void func_int(int sig)
{ /* trata sinal */  }
```

```
main()
{  ...
   signal(SIGINT, func_int);
   puts("func_int instalada p/ SIGINT\n");
...
}
```

Funcionamento dos sinais

- **Os detalhes do funcionamento dos sinais têm mudado com as versões do sistema… ☹**

Funcionamento dos sinais

- **Os detalhes do funcionamento dos sinais têm mudado com as versões do sistema… ☹**
  - **o *handler* é mantido após entregue cada sinal ou não?**

Funcionamento dos sinais

- **Os detalhes do funcionamento dos sinais têm mudado com as versões do sistema… ☹**
  - **o *handler* é mantido após entregue cada sinal ou não?**
  - **podem-se perder sinais enquanto se tratam outros… interrompem o primeiro… ou são guardados?**

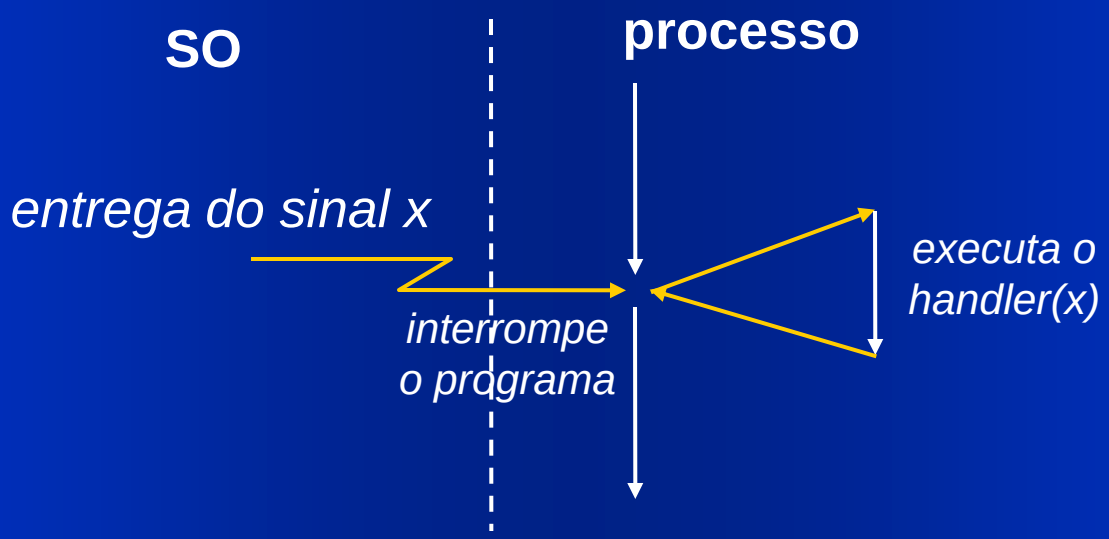
Funcionamento dos sinais

- **Os detalhes do funcionamento dos sinais têm mudado com as versões do sistema… ☹**
  - **o *handler* é mantido após entregue cada sinal ou não?**
  - **podem-se perder sinais enquanto se tratam outros… interrompem o primeiro… ou são guardados?**
  - **os sinais interrompem a execução das chamadas ao sistema que bloqueiam o processo invocador?**

Funcionamento dos sinais

- **Os detalhes do funcionamento dos sinais têm mudado com as versões do sistema… ☹**
  - **o *handler* é mantido após entregue cada sinal ou não?**
  - **podem-se perder sinais enquanto se tratam outros… interrompem o primeiro… ou são guardados?**
  - **os sinais interrompem a execução das chamadas ao sistema que bloqueiam o processo invocador?**
- **Foi definida uma norma com uma nova interface: `sigaction`**
  - **esta permite definir o comportamento pretendido**
  - **os sinais podem incluir mais informação**

Tratamento de sinais (recepção)



- **O *handler* é definido pela chamada ao SO:**

```
int sigaction(int signo, const struct sigaction *act,
                                 struct sigaction *oact);

 struct sigaction {
   void (*sa_handler)(int);   -- acção a efectuar
   sigset_t sa_mask;          -- + sinais a bloquear durante tratamento
   int sa_flags;              -- comportamento após sinal
   void (*sa_sigaction)(int, siginfo_t *, void *);
 }
```

Fiabilidade dos sinais no Unix

- **Primeiras versões dos sinais Unix (anteriores a 4.2 BSD e SVR3) não eram confiáveis:**
  - **Por cada sinal de um certo tipo entregue a um processo, a acção de tratamento passava a ser a pré-definida pelo SO**
  - **era preciso 're-instalar' o *handler* de cada vez que ocorria um sinal desse tipo.**
  - **ocorriam erros dependentes do tempo**
    - ***race-conditions*** **(corridas)**

Fiabilidade dos sinais no Unix (2)

- **Versões modernas do Unix e a norma POSIX (`sigaction`):**
  - **o *handler* de um sinal mantém-se definido até que o programa o mude**
  - **pode-se bloquear/mascarar a entrega de sinais (pelo seu tipo), p.ex.:**
    - **durante zonas críticas do programa,**
    - **durante a execução dos *handlers***

  Mas … a programação com sinais Unix continua a ser muito complexa e dependente das versões do sistema

Outra dificuldade

- **Na maioria das realizações:**
  - **se ocorrerem dois ou mais sinais de um mesmo tipo, sucessivamente, antes de o SO entregar o primeiro deles,**
  - **então: não há garantia alguma de que todos os sinais sejam entregues.**
- **O que é garantido pela norma POSIX:**
  - **Permite que seja entregue 1 ou mais vezes…**

Sinais durante chamadas ao SO

- **O que acontece quando é gerado um sinal destinado a um processo, que invocou uma chamada ao SO e ainda não retornou ao programa utilizador?**
- **em geral, o sinal só é entregue no fim, isto é, logo que se dá o retorno ao processo**
- **mas se a chamada ao SO bloquear?**
  - **`read/write em pipe, wait`**
- **o comportamento é diferente!**

Porquê?

Sinais durante chamadas bloqueadas

1. **tratamento:** o processo, ainda que bloqueado, é "interrompido" e a acção definida para o tratamento do sinal é efectuada;
2. **retorno:** em vez de retornar ao estado bloqueado, no interior da chamada onde estava, esta **pode ser abortada, por opção**:
   - retorna com erro (-1), e `errno` afectada a `EINTR`, que indica que foi interrompida por um sinal.

Sinais durante chamadas ao SO (3)

O tratamento após a interrupção da chamada ao SO pode ser:

Sinais durante chamadas ao SO (3)

O tratamento após a interrupção da chamada ao SO pode ser:

- **repetir a chamada, voltando potencialmente a bloquear-se, ou …**

Sinais durante chamadas ao SO (3)

O tratamento após a interrupção da chamada ao SO pode ser:

- **repetir a chamada, voltando potencialmente a bloquear-se, ou …**
- **prosseguir a execução deixando a decisão de qual acção a tomar à responsabilidade do programa**
- **(definida por uma opção em *sigaction()*).**

Exemplo: aguardar um intervalo de tempo

- **A função `sleep(T)`**
  - **bloqueia o processo até que decorra um intervalo de T segundos.**
- **Se quisermos que o processo seja avisado de que decorreu um certo intervalo de tempo, mas sem ter de ficar bloqueado à espera?**
  - **recorre-se a um sinal de tipo: SIGALRM**

Exemplo de sinal de alarme

- **Chamada ao SO:**

```
int alarm(int nseg)
```

  - **gera um sinal de tipo SIGALRM, que é entregue ao próprio processo, após decorridos `nseg` segundos…**

Exemplo de sinal de alarme

- **Chamada ao SO:**

```
int alarm(int nseg)
```

  - **gera um sinal de tipo SIGALRM, que é entregue ao próprio processo, após decorridos `nseg` segundos…**
  - **devolve o número de segundos que faltam até algum alarme anterior ocorrer ou devolve 0 se nenhum alarme estiver activo:**

Exemplo de sinal de alarme

- **Chamada ao SO:**

  `int alarm(int nseg)`

  - **gera um sinal de tipo SIGALRM, que é entregue ao próprio processo, após decorridos `nseg` segundos…**
  - **devolve o número de segundos que faltam até algum alarme anterior ocorrer ou devolve 0 se nenhum alarme estiver activo:**
  - **`alarm(0)`: desliga o alarme**

Exemplo de sinal de alarme (2)

- **O processo é notificado da ocorrência do alarme:**
  - **se o programa definiu um *handler*, este é invocado e depois o programa interrompido continua**
- **isto permite invocar acções periodicamente**
- **permite realizar *TIME-OUT*s:**
  - **pode interromper algumas chamadas ao sistema, por exemplo: se o processo estava bloqueado em `read` de um *pipe* vazio, a entrega do sinal de alarme permite desbloqueá-lo ao fim de um TIME-OUT.**

Exemplo: mySleep

- Como obter o efeito do *sleep*?

```
void alarmHandler(int signo)
{ return; }
```

```
void my_Sleep( int s )
{   signal(SIGALRM, alarmHandler);
    alarm(s);
    pause();
    signal(SIGALRM, SIG_IGN);
}
```

Exemplo: `read` com *time-out*

Assumindo *handler* definido para SIGALRM:

```
alarm(5);
n = read( 0 , buff, SZ );
if ( n == -1 && errno == EINTR )
  /* houve time-out */
   …
else
   alarm(0);
   …
```

fork/exec e os sinais

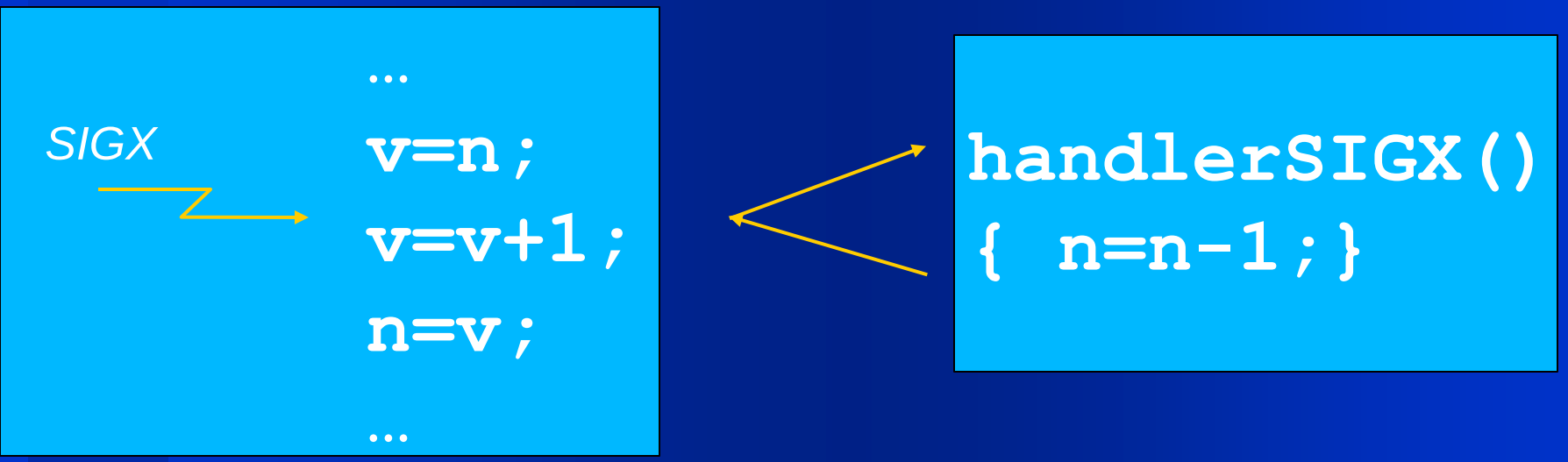
- **Após exec():**
  - **todos os sinais que tiverem um *handler* definido pelo programa, passam à acção pré-definida pelo SO;**
  - **todos os outros sinais ficam com a mesma acção antes definida.**

fork/exec e os sinais

- **Após exec():**
  - **todos os sinais que tiverem um *handler* definido pelo programa, passam à acção pré-definida pelo SO;**
  - **todos os outros sinais ficam com a mesma acção antes definida.**
- **Após fork():**
  - **o filho herda as definições dos sinais do pai;**
  - **mas não recebe um SIGALRM que o pai tenha pedido**

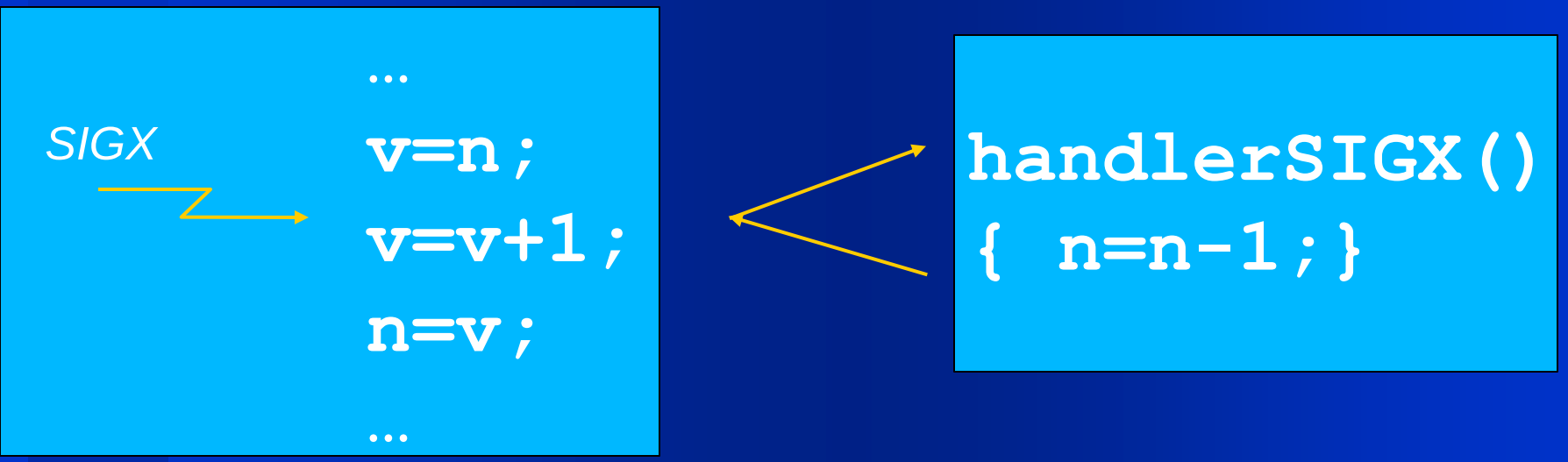
Problemas de concorrência …

- **Se não há controlo sobre quando um sinal ocorre, podemos ter problemas. Exemplo:**



Problemas de concorrência ...

- **Se não há controlo sobre quando um sinal ocorre, podemos ter problemas. Exemplo:**



- **Este é um exemplo no mesmo processo, devido a um sinal. Este tipo de problemas pode ocorrer sempre que há processos concorrentes partilhando recursos...**

Signal sets (conjuntos de sinais)

Tipo sigset_t

representa cada sinal por um bit

Funções

- **sigemptyset – tudo a 0**
- **sigfillset – tudo a 1**
- **sigaddset – um sinal específico a 1**
- **sigdelset – um sinal específico a 0**

Signal sets (conjuntos de sinais)

Tipo sigset_t

representa cada sinal por um bit

Funções

- **sigemptyset – tudo a 0**
- **sigfillset – tudo a 1**
- **sigaddset – um sinal específico a 1**
- **sigdelset – um sinal específico a 0**

```
int sigemptyset(sigset_t  *set),
int sigaddset(sigset_t  *set, int signo);
```

```
int sigaction(int signo,
              const struct sigaction    *act,
                    struct    sigaction      *oact);
```

act – as acções a efectuar

oact – os valores correntes antes definidos

```
int sigaction(int signo,
              const struct sigaction    *act,
                 struct    sigaction      *oact);
```

act – as acções a efectuar

oact – os valores correntes antes definidos

```
struct sigaction{
        void  (*sa_handler)(int); -- a acção a efectuar
        sigset_t     sa_mask;
        -- sinais a bloquear durante o tratamento
        int     sa_flags;       -- opções para tratar
        void (*sa_sigaction)(int, siginfo_t *, void *);
        -- opção para o handler  se SA_SIGINFO
}
```

sa_handler:

SIG_DFL – instalar a acção *by default*

SIG_IGN – ignorar a ocorrência do sinal deste tipo

endereço de um *handler* (nome da função)

sa_mask: especifica um conjunto de sinais a bloquear durante a execução do *handler*, além do tipo do sinal a tratar

sa_flags: modifica o comportamento após o tratamento:

-- sa_flags = SA_RESETHAND -- reset SIG_DFL

-- sa_flags = SA_SIGINFO após a recepção do sinal a estrutura siginfo_t tem informação adicional do sinal

-- sa_flags = SA_RESTART repete a execução de uma chamada ao SO interrompida por um sinal, sem dar erro

sa_sigaction: -- outra opção para especificar o *handler*

Bloqueio de sinais

```
int sigprocmask(int how,
                const sigset_t    *set,
                sigset_t    *oset);
```

how: SIG_SETMASK bloqueia os sinais de *set*

SIG_UNBLOCK desbloqueia-os

SIG_BLOCK junta os sinais de *set* à máscara corrente

oset: obtém a máscara corrente de sinais bloqueados

int kill(pid_t pid, int sig);

O emissor deve ter o mesmo userID do receptor, excepto no caso do *superuser*

int kill(pid_t pid, int sig);

O emissor deve ter o mesmo userID do receptor, excepto no caso do *superuser*

pid = 0 envia a todos processos do mesmo grupo do emissor

int kill(pid_t pid, int sig);

O emissor deve ter o mesmo userID do receptor, excepto no caso do *superuser*

pid = 0 envia a todos processos do mesmo grupo do emissor

pid > 0 envia ao processo de pid dado

int kill(pid_t pid, int sig);

O emissor deve ter o mesmo userID do receptor, excepto no caso do *superuser*

pid = 0 envia a todos processos do mesmo grupo do emissor

pid > 0 envia ao processo de pid dado

pid = -1 envia aos processos com o mesmo userID do emissor

se for o superuser: envia a todos os processos

int kill(pid_t pid, int sig);

O emissor deve ter o mesmo userID do receptor, excepto no caso do *superuser*

pid = 0 envia a todos processos do mesmo grupo do emissor

pid > 0 envia ao processo de pid dado

pid = -1 envia aos processos com o mesmo userID do emissor

se for o superuser: envia a todos os processos

pid <0 mas <> -1: envia aos processos do grupo cujo gid é igual ao |pid|

int raise(int sig);

Envia um sinal ao próprio processo invocador

int alarm(int secs);

Envia sinal SIGALRM ao próprio, ao fim de secs segundos

é desligado com alarm(0);

int pause(void);

Aguarda, bloqueado, qualquer tipo de sinal

Se é para tratar: devolve -1 e errno = EINTR