I2180
LINUX :
APPELS SYSTÈME

LES SIGNAUX

SYNCHRONICITÉ

- **Synchrone** = l'exécution d'un programme respecte un séquençage temporel de son flux algorithmique

- **Asynchrone** = suite à un événement externe au processus (p.ex. émission d'un signal), le système interrompt l'exécution normale du programme afin qu'il exécute un traitement particulier.

LES SIGNAUX UNIX

- Signal = interruption logicielle asynchrone
  $\Rightarrow$ exécution du programme suspendue
- On parle d' « émission » et de « réception » d'un signal.
- Un processus peut choisir de se comporter de trois manières différentes lorsqu'il reçoit un signal :
  1. ignorer le signal,
  2. effectuer l'action par défaut,
  3. traiter le signal avec un gestionnaire de signal.
- Cf. **`man 7 signal`**
  (`kill -l` $\rightarrow$ affichage des signaux définis par le système)

ACTIONS PAR DÉFAUT

- Une action par défaut est définie pour chaque signal:
  - Ignore (ignorer le signal)
  - Terminate (terminer le processus)
  - Core (créer un fichier *core* et terminer le processus)
  - Stop (arrêter le processus)
  - Continue (continuer le processus s'il est arrêté)
- Généralement : Terminate

  (justification: si un processus n'est pas préparé à gérer la réception d'un signal particulier, c'est qu'il ne doit pas le recevoir! S'il le reçoit quand même, c'est le signe probable d'une anomalie grave et il est donc plus prudent de terminer le processus.)

ARMEMENT D'UN SIGNAL

- La fonction appelée lors de la réception d'un signal se nomme *gestionnaire de signal* ou *signal handler*.
- Lorsqu'un processus enregistre un gestionnaire de signal, on dit qu'il « arme » le handler de signal. Ce handler sera exécuté lors de l'interception du signal par le processus.
- Un handler peut être associé à un signal grâce aux appels système : ~~`signal` et~~ `sigaction`.

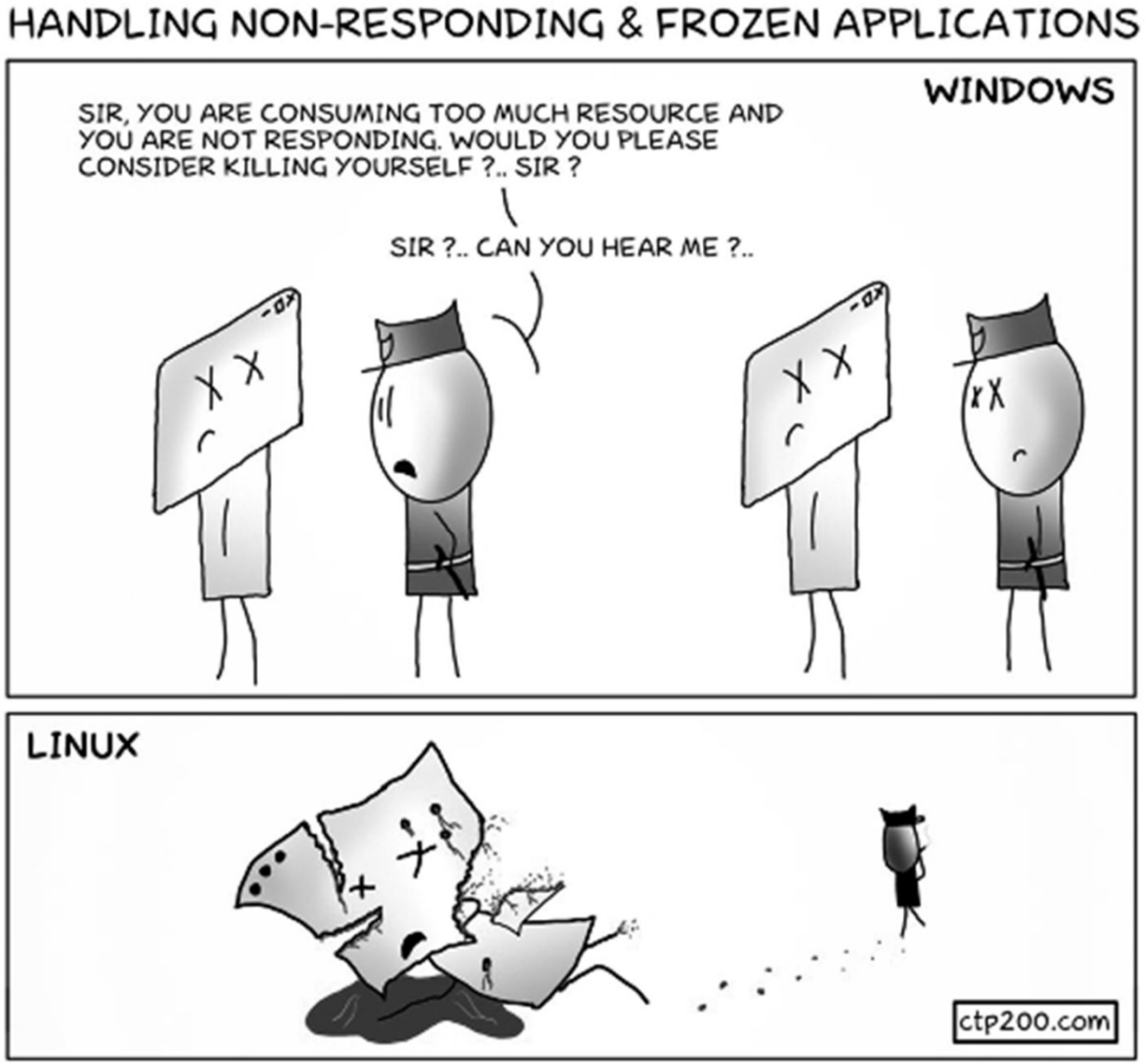
EXEMPLE DE SIGNAL : SIGINT

- SIGINT : signal d'interruption clavier,
  défini dans <signal.h> (cf. man 7 signal)
- Valeur = 2 (dépendant de l'architecture)
- Action par défaut : terminaison (propre) du processus
- Émission du signal SIGINT :
  - Ctrl-C au clavier
  - commande shell : kill -SIGINT pid *ou* kill -2 pid
  - appel système : kill(pid, SIGINT)
- SIGINT peut être intercepté (i.e. dérouté vers une fonction *handler*), ignoré ou bloqué ; sinon action par défaut (terminaison propre du processus)

EXEMPLE DE SIGNAL : SIGKILL

- SIGKILL : signal « KILL » défini dans <signal.h>
  (cf. man 7 signal)
- Valeur = 9 (dépendant de l'architecture)
- Action par défaut : terminaison (brutale) du processus
- Émission du signal SIGKILL :
  - commande shell : kill -SIGKILL pid *ou* kill -9 pid
  - appel système : kill(pid, SIGKILL)
- SIGKILL ne peut être ni intercepté, ni ignoré ou bloqué
  $\Rightarrow$ une fois un SIGKILL reçu, la « mort » du processus est inévitable (terminaison brutale du processus)

kill -SIGKILL pid



AUTRES SIGNAUX

SIGNAL	DESCRIPTION
SIGABRT	signal émis lorsqu'un programme rencontre un problème forçant son arrêt (génère un *core dump*)
SIGFPE	signal émis lors d'une erreur arithmétique (*floating point exception*), forçant l'arrêt du processus (génère un *core dump*)
SIGSEGV	erreur de protection mémoire (*segmentation fault*) i.e. un processus tente d'écrire en dehors de son espace d'adressage
SIGUSR1 SIGUSR2	signaux *user*, personnalisables
SIGPIPE	signal émis lorsqu'on envoie des données sur un *pipe* dont l'autre bout est fermé en lecture (*broken pipe*)
SIGALRM	signal utilisé pour programmer des temporisations
SIGTERM	idem SIGINT mais pas de combinaison de touches pour le générer
SIGCHLD	signal émis lorsqu'un processus fils est mort ou a été arrêté
SIGCONT	signal provoquant le redémarrage d'un processus temporairement arrêté
SIGSTOP	signal provoquant l'arrêt temporaire d'un processus ; ne peut être ignoré
SIGTSTP	signal provoquant l'arrêt temporaire d'un processus (Ctrl-Z) ; peut être ignoré

- Appel système historique d'UNIX pour assurer la programmation du gestionnaire de signaux

- MAIS problèmes de portabilité, désarmement automatique de la fonction handler (⇒ risque d'exécution de l'action par défaut avant le réarmement du handler), risque d'interruption du gestionnaire par l'arrivée d'un nouveau signal, etc.)

  ➔ appel système **`signal`** considéré comme non fiable !

POSIX

`sigaction`

- Programmation (fiable) d'un gestionnaire de signal
- Un handler armé par `sigaction` le reste jusqu'à ce qu'un autre handler soit armé pour le même signal.

sigaction

```
#include <signal.h>

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

- Où: signum : numéro du signal ∈ [1,NSIG]

  act : nouveau comportement à adopter en cas de réception du signal signum

  oldact : sauvegarde de l'ancienne action

- Renvoie 0 si réussi ; -1 si échec

- Lors d'un *fork*, les processus père et fils partagent les mêmes actions et gestionnaires associés aux signaux

  → pour éviter de perdre un signal entre père et fils, armer le signal avant le *fork*.

Structure sigaction

```
struct sigaction {
   void (*sa_handler)(int);
   void (*sa_sigaction)(int, siginfo_t*, void*);
   sigset_t sa_mask;
   int sa_flags;
   void (*sa_restorer)(void);
};
```

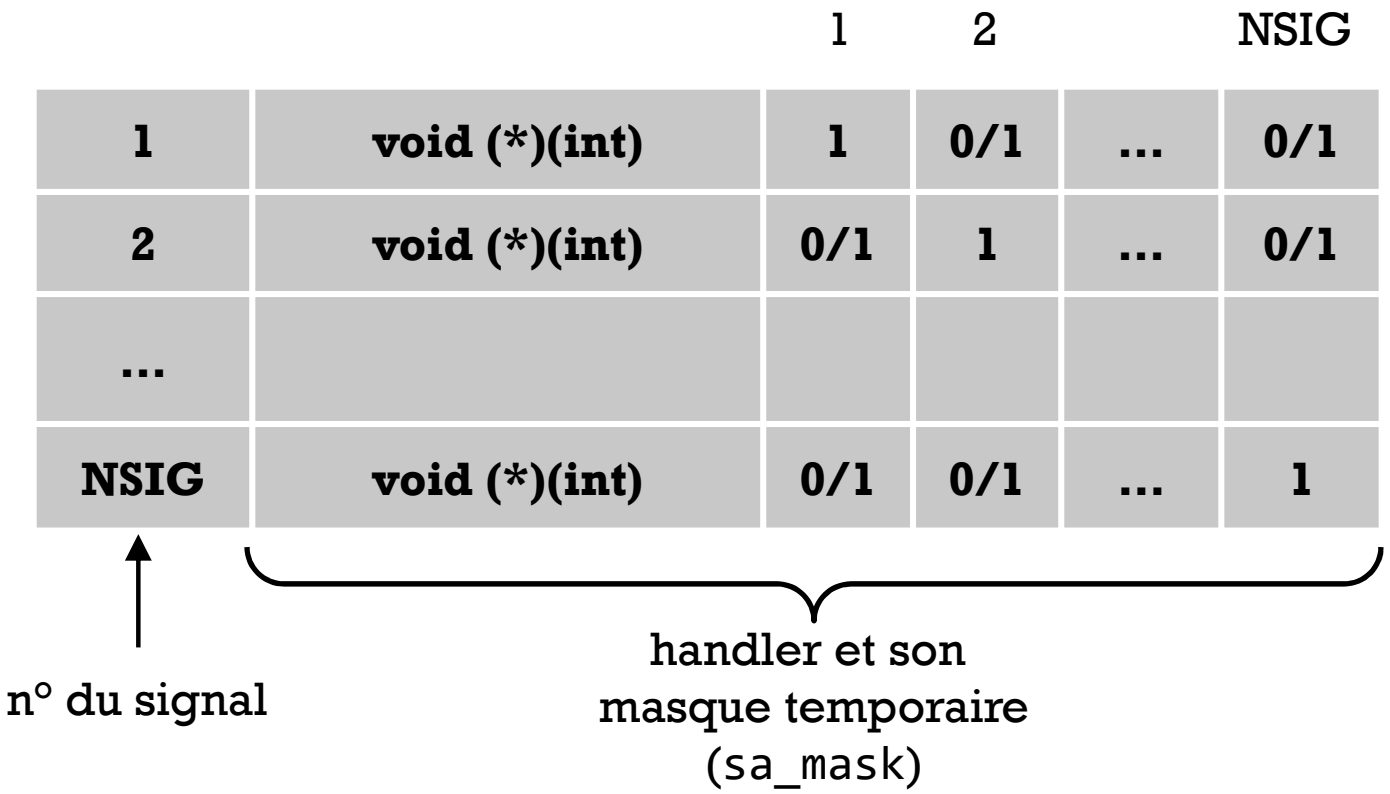
Où : sa_handler : gestionnaire de signal (avec le numéro du signal qui l'a déclenché en paramètre) ou actions SIG_IGN / SIG_DFL

sa_mask : ensemble des signaux à bloquer pendant l'exécution du handler (y compris le signal lui-même)

sa_flags : options de configuration du gestionnaire de signaux (cf. man sigaction) → 0 pour comportement par défaut

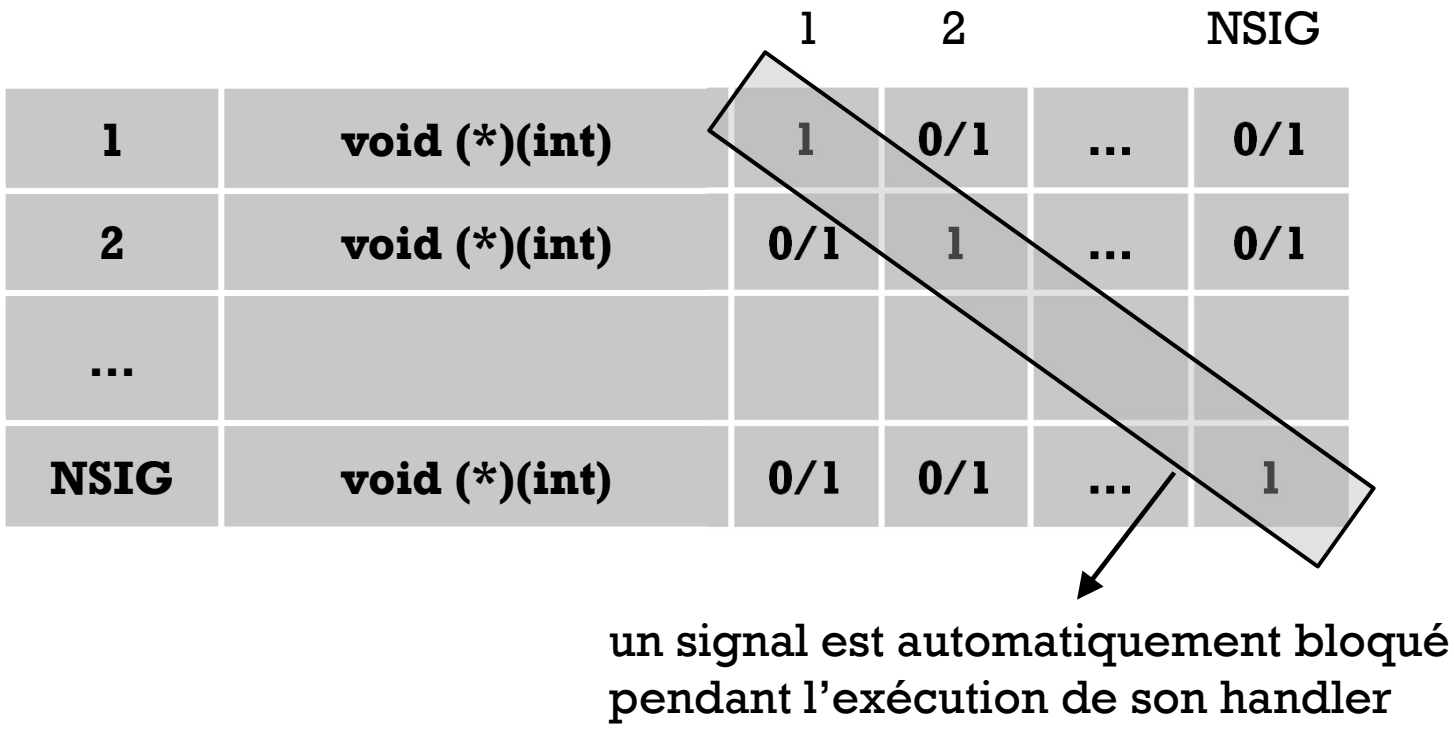
Masque de sigaction (sa_mask)

Gestionnaires de signaux d'un processus:



Masque de sigaction (sa_mask)

Gestionnaires de signaux d'un processus:



kill

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

- Envoyer un signal à un processus
- Renvoie 0 si succès ; -1 si échec
- Où: pid : numéro du processus cible du signal

  sig : numéro du signal à envoyer (sig ∈ [1,NSIG])

pause

```
#include <unistd.h>

int pause(void);
```

- Mise en attente de réception d'un signal quelconque
- 3 cas selon le signal reçu :
  - Le processus continue à attendre sans *return* de *pause* si l'action associée au signal reçu est « *ignore* ».
  - Fin du processus sans *return* de *pause* si l'action associée au signal reçu est « *terminate* ».
  - Si un handler a été armé pour le signal reçu, *pause* renverra -1 avec errno = EINTR après l'exécution de ce handler puis le processus reprendra son exécution

EXEMPLE

- Cf. exemple1.c

- Compilation avec l'option:

  `-D_POSIX_C_SOURCE`

  (cf. man 7 feature_test_macros)

TRAITEMENT PAR UN HANDLER

Pour pouvoir être interrompu sans risque de provoquer un état incohérent, un gestionnaire de signaux doit être aussi concis que possible et appliquer une des trois règles suivantes :

- appeler uniquement des fonctions « asynchronous(-signal)-safe » (ou *atomiques*), i.e. des opérations durant lesquelles il n'est pas possible d'être interrompu : l'opération est effectuée d'un seul tenant
  (cf. 'man 7 signal' pour la liste des fonctions *async-signal-safe*)
- accéder uniquement à des variables de type « atomic_t »
  (cf. stdatomic.h)
- bloquer la réception des signaux durant son exécution
  (cf. le masque de `sigaction` ou l'appel système `sigprocmask`)