I2180 PROGRAMMATION 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 : affichage de tous les 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 est appelée *gestionnaire de signal* ou *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 : 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 $\Rightarrow$ terminaison propre du processus

EXEMPLE : 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 ni être intercepté, ni ignoré ou bloqué
  $\Rightarrow$ une fois un SIGKILL reçu, la « mort » du processus est inévitable (terminaison brutale du processus)

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 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

signal

- Appel système historique d'UNIX pour assurer la programmation du gestionnaire de signaux
- MAIS problèmes de portabilité, désarmement automatique de la fonction gestionnaire de signal (⇒ risque d'exécution de l'action par défaut avant le réenregistrement du handler), risque d'interruption du gestionnaire par l'arrivée d'un nouveau signal, etc.)

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

sigaction

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

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.

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 ou actions SIG_IGN et 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)

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 :
  - Pas de retour du processus de la fonction *pause* pour un signal ignoré.
  - Fin du processus sans retour de *pause* si l'action associée au signal reçu est la terminaison du processus.
  - Si un handler a été armé pour le signal reçu, *pause* renverra -1 avec errno = EINTR après l'exécution du gestionnaire puis le processus reprendra son exécution

EXEMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>

void sigusr1_handler (int sig) {
    printf("[Processus %d] Signal SIGUSR1 recu\n", getpid());
    exit(10);
}
```

EXEMPLE

```
int main () {
    pid_t pid;
    int statut;

    // initialisation des champs de la structure sigaction à 0
    struct sigaction newact = {{0}};
    // définition du handler de SIGUSR1
    newact.sa_handler = sigusr1_handler;
    // armement du signal SUGUSR1
    sigaction(SIGUSR1, &newact, NULL);
```

EXEMPLE

```
if ((pid=fork())==0) {
    /* processus fils */
    printf("Je suis le fils de PID %d\n", getppid());
    pause();
    exit(0);
}
/* processus parent */
printf("Je suis le pere de PID %d - ", pid);
printf("envoi du signal SIGUSR1 à mon fils\n");
kill(pid,SIGUSR1);
pid=waitpid(pid,&statut,0);
printf("Statut de mon fils %d: %d\n", pid, WEXITSTATUS(statut));
}
```

EXEMPLE

- Compilation avec l'option: **–D_DEFAULT_SOURCE**

  (cf. man 7 feature_test_macros)

- **OUPUT :**

```
Je suis le pere de PID 31234 - envoi du signal
SIGUSR1 à mon fils
[Processus 31234] Signal SIGUSR1 recu
Statut de mon fils 31234 : 10
```