Les signaux

Cyril Rabat
cyril.rabat@univ-reims.fr

M2CCI - Système d'exploitation

2008-2009

Cours n°7

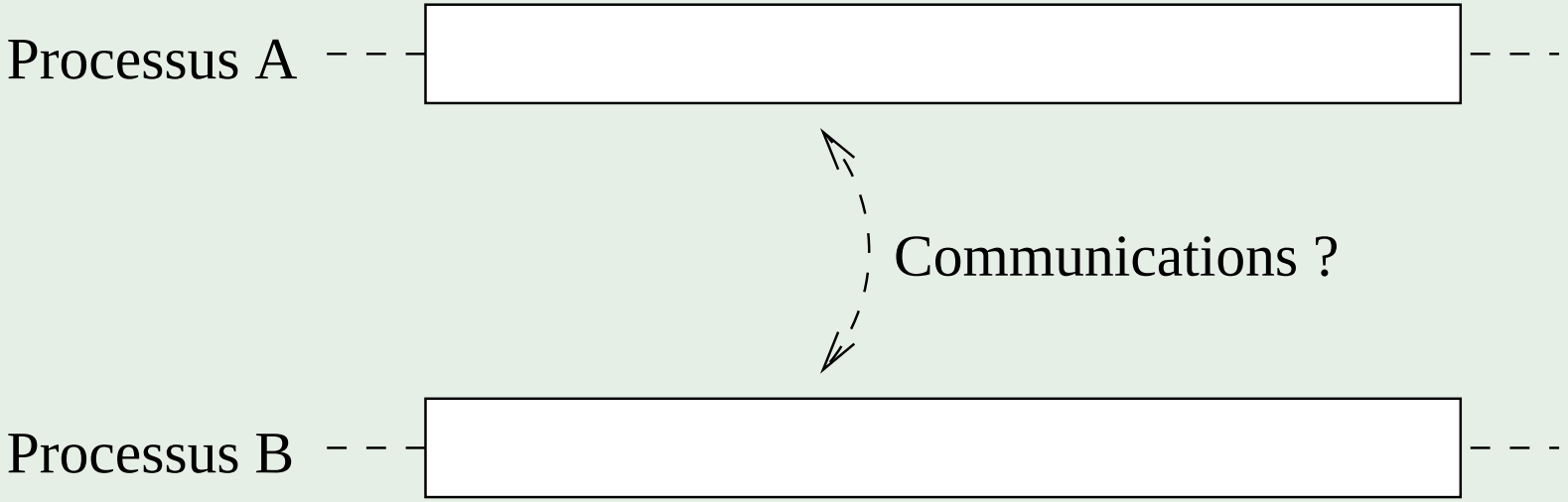
Les signaux (signaux particuliers, méthodes C).

Table des matières

Introduction

Problématique : comment communiquer entre deux processus ?

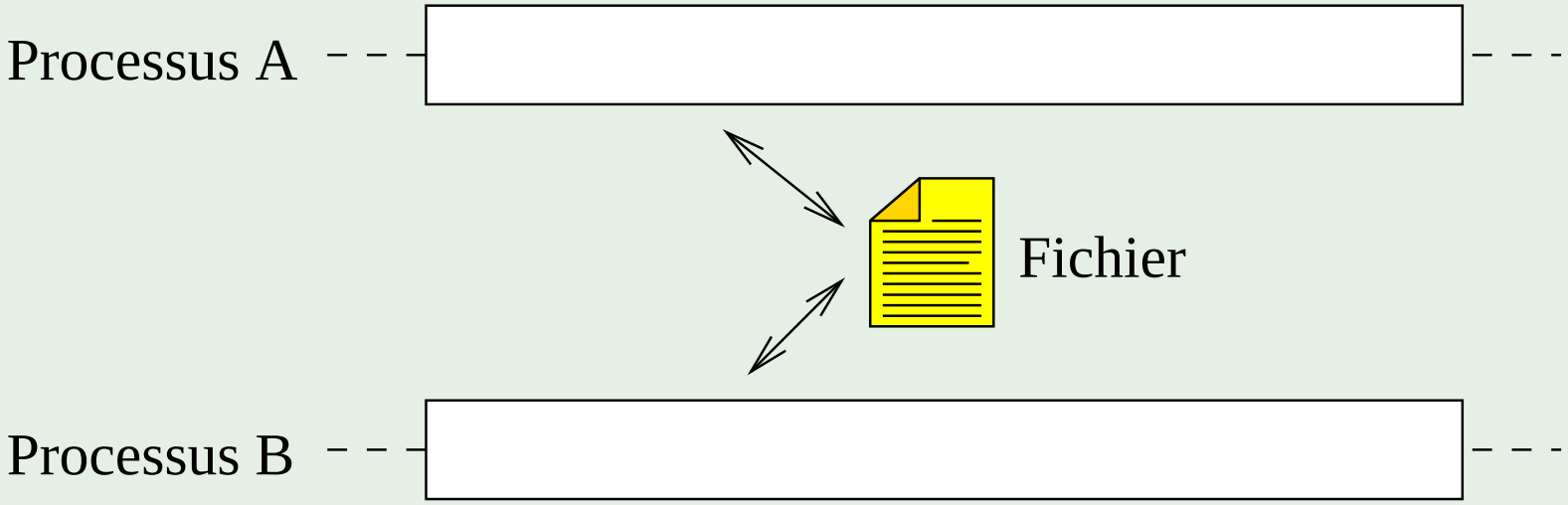
- Comment communiquer entre des processus indépendants ?



Introduction

Problématique : comment communiquer entre deux processus ?

- Idée : passer par un fichier partagé.



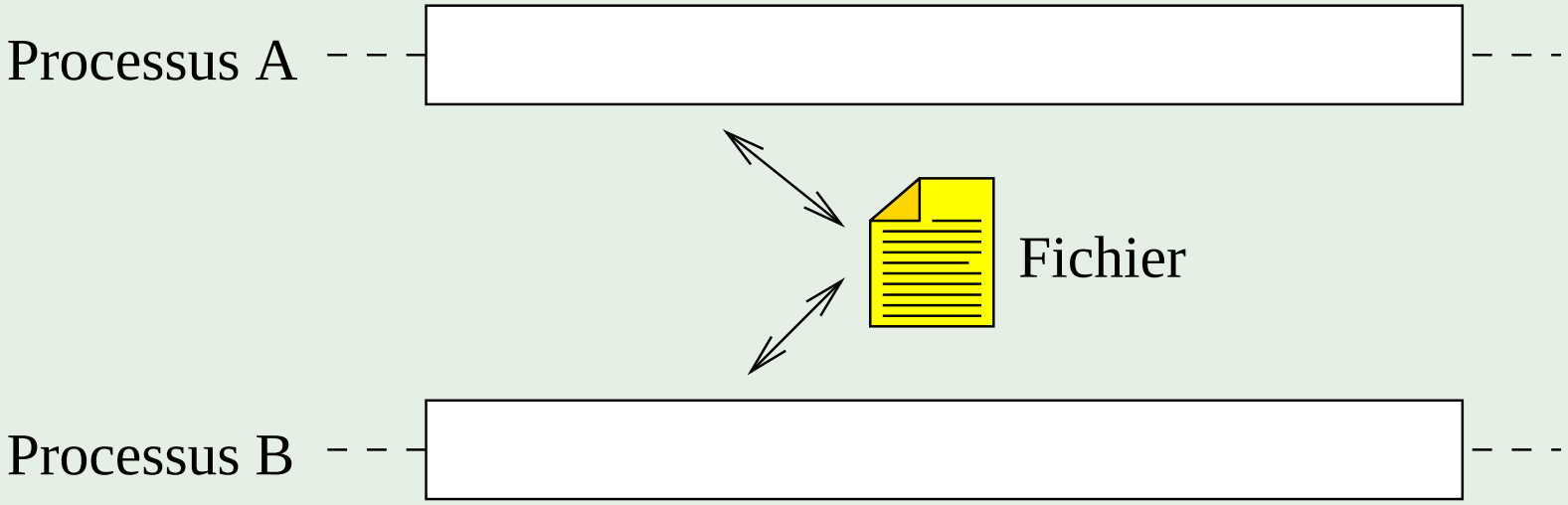
Problème

Il faut gérer les problèmes de concurrence sur les accès au fichier !

Introduction

Problématique : comment communiquer entre deux processus ?

- Idée : passer par un fichier partagé.



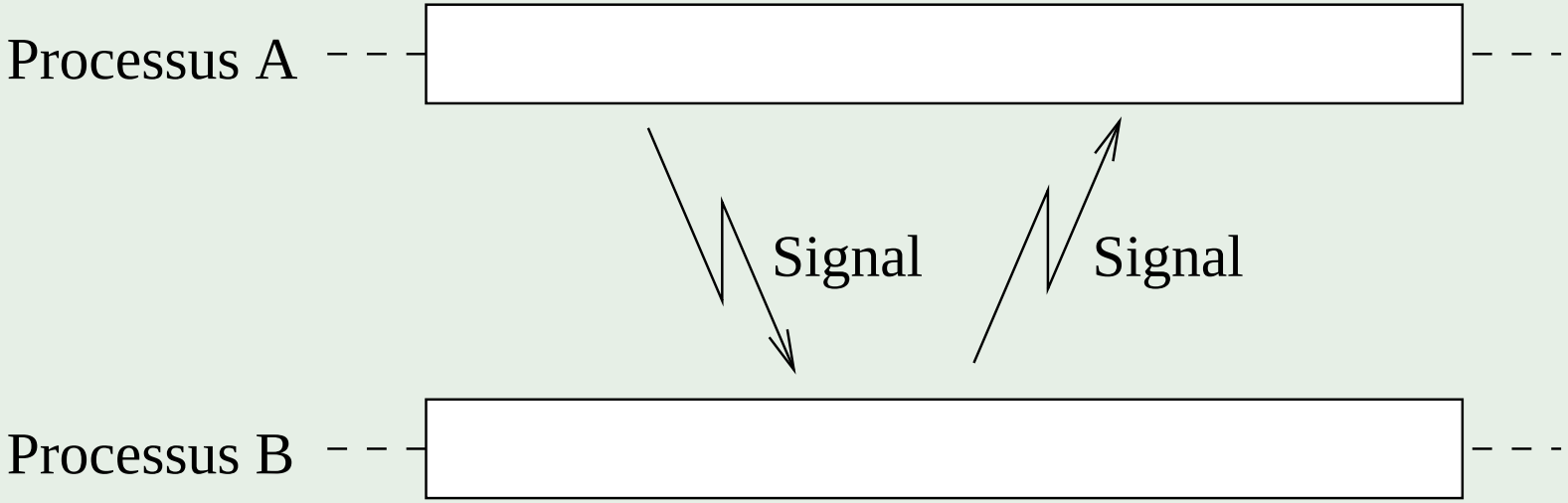
Problème

Il faut gérer les problèmes de concurrence sur les accès au fichier !

Introduction

Problématique : comment communiquer entre deux processus ?

- Utilisation des signaux.



Les signaux

Définition : un signal

Un signal est une interruption logicielle asynchrone.

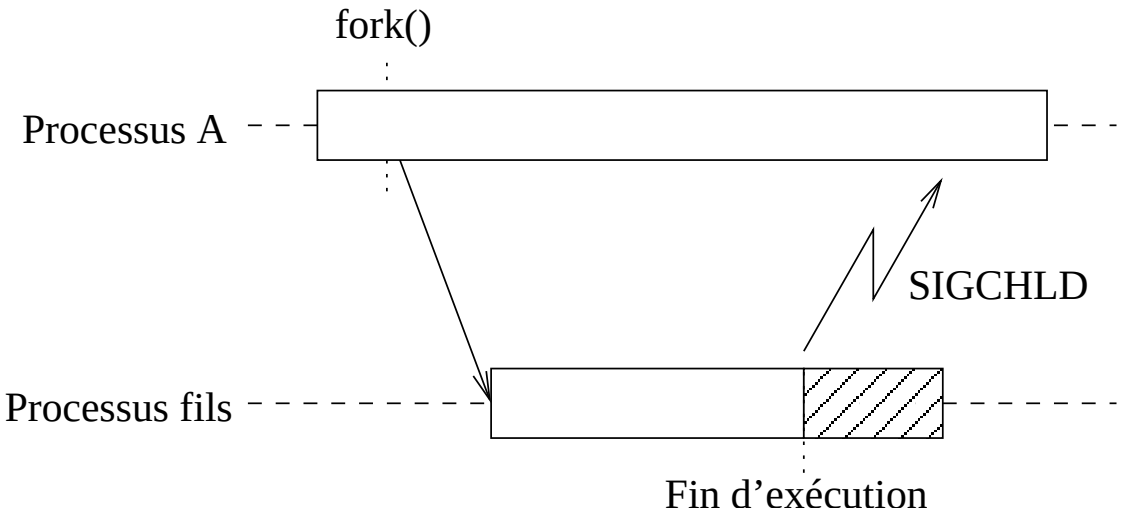
- Les signaux permettent de communiquer entre processus ou entre le système et les processus ;
- Sous Windows : les "*events*" ;
- Il y a 4 environnements de signaux qui cohabitent sous Unix :
  - System V non-fiable ;
  - BSD 4.x ;
  - System V fiable (pour *reliable*) (SVR3) ;
  - POSIX.

Attention

Il est possible de faire appel précisément à l'un de ces environnements : c'est déconseillé (problème de portabilité du code !)

La fin d'exécution d'un fils

- Lorsqu'un fils termine son exécution, un signal SIGCHLD est envoyé au père ;
- Le fils devient un processus zombie ;
- Lorsque le père capte le signal (fonction `wait`), le fils est définitivement détruit.

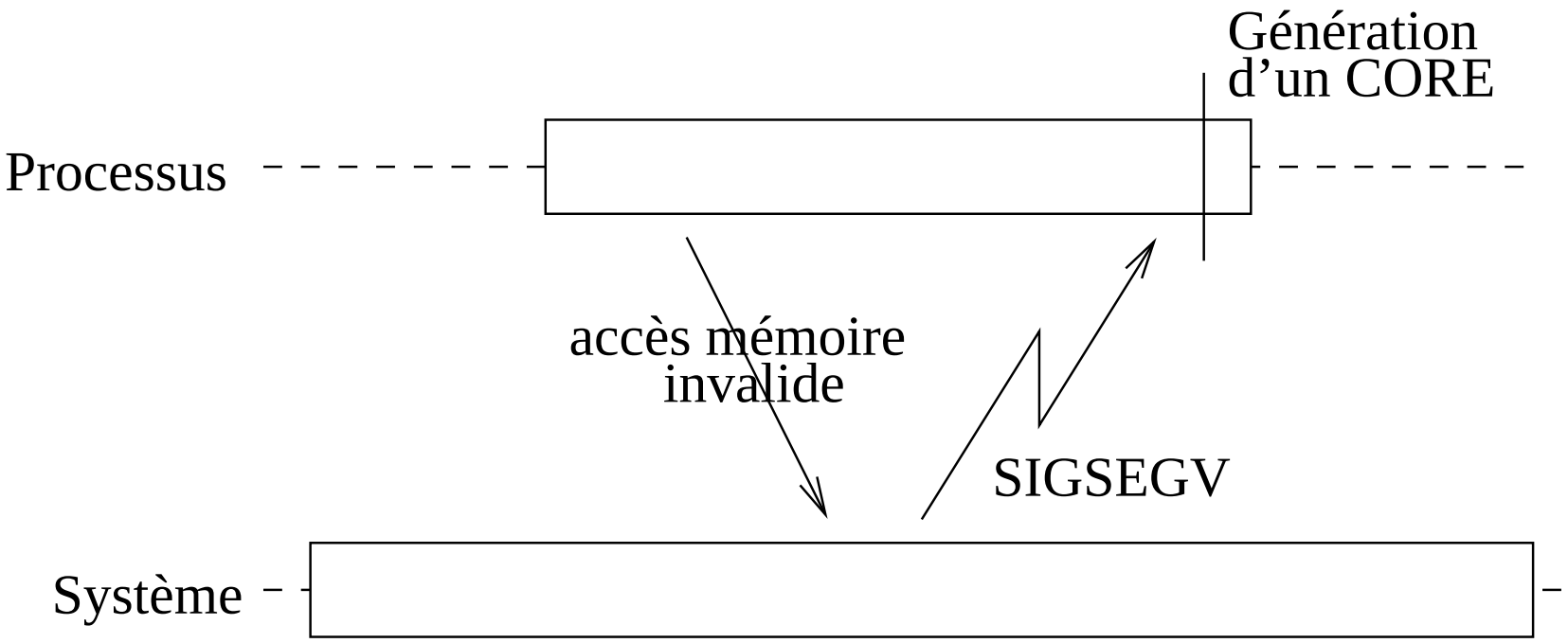


Attention

Une application qui crée des processus (exemple d'un serveur multi-clients) doit capturer les signaux SIGCHLD pour éviter une saturation mémoire.

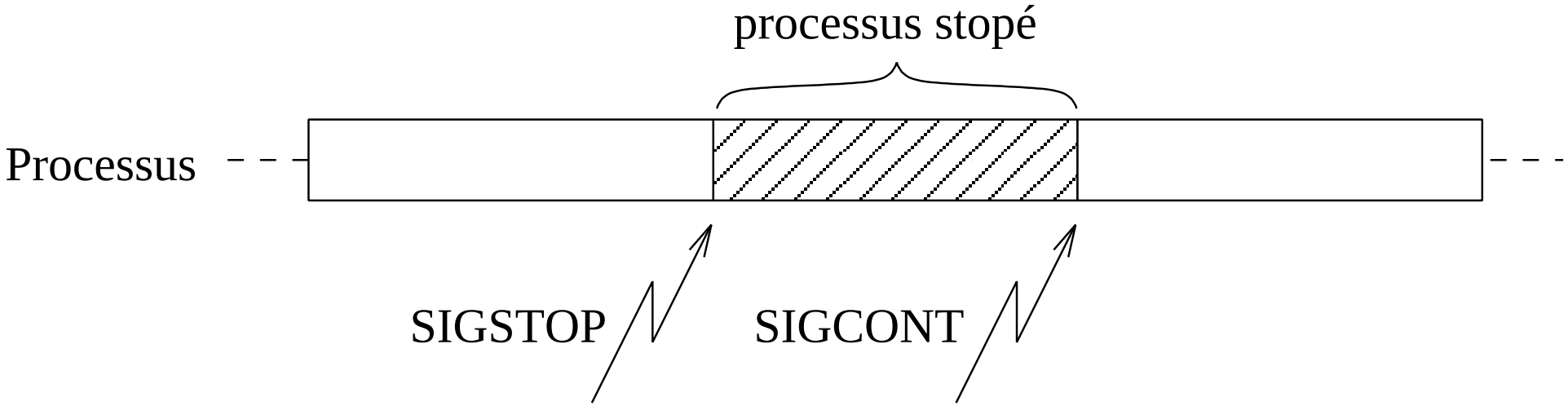
Violation d'accès mémoire

- Si un processus accède à une zone mémoire *interdite*, le système d'exploitation empêche cet accès :
  - Envoi d'un signal SIGSEGV au processus fautif ;
  - Une copie mémoire du processus est réalisée dans un fichier (CORE) ;
  - Le processus est tué.



Stopper un processus (et redémarrage)

- Il est possible de stopper l'exécution d'un processus à l'aide du signal "`SIGSTOP`" ;
- Le processus reste en mémoire mais ne consomme plus de CPU ;
- Pour continuer son exécution, il faut lui envoyer le signal "`SIGCONT`".



Envoyer des signaux depuis le terminal

- Depuis le terminal, avec la commande `kill` :

```
kill -sigstop 1234
```

  $\Longrightarrow$ Le processus dont l'identifiant est 1234 est stopé.
- Des raccourcis clavier permettent aussi d'envoyer des signaux au processus courant :
  - `CTRL+C` : envoie le signal SIGINT au processus courant ;
  - `CTRL+D` : envoie le signal SIGHUP au processus courant ;
  - `CTRL+Z` : envoie le signal SIGSTOP au processus courant ...

Appels système et fonctions C pour envoyer des signaux

- `int kill(pid_t pid, int sig)`
  $\Longrightarrow$ envoie le signal "`sig`" au processus dont l'identifiant est "`pid`"
- `int raise(int sig)`
  $\Longrightarrow$ idem mais envoie le signal au processus courant
- `unsigned int alarm(unsigned int nb_sec)`
  $\Longrightarrow$ envoie le signal "`SIGALRM`" au processus courant dans "`nb_sec`" secondes : retourne le nombre de secondes restantes depuis le dernier appel à `alarm` ou 0
- `int pause(void)`
  $\Longrightarrow$ stoppe le processus jusqu'à réception d'un signal.

Liste des signaux et actions par défaut

Signal	*Actions par défaut*	*Description*
SIGHUP	Termine le processus	Fin de connexion
SIGINT	Termine le processus	Interruption
SIGQUIT	Termine avec un core	Interruption forte
SIGKILL	Termine le processus	Interruption très forte
SIGALRM	Ignoré	Interruption horloge
SIGFPE	Termine le processus	Erreur arithmétique (division par 0)
SIGCHLD	Ignoré	Fin d'un fils
SIGUSR1	Termine le processus	Signal utilisateur
SIGUSR2	Termine le processus	Signal utilisateur
SIGPIPE	Termine le processus	Écriture dans un tube sans lecteur ou tube cassé
SIGSEGV	Termine avec un core	Violation mémoire
SIGSTOP	Stoppe le processus	Le processus est stoppé
SIGCONT	Ignoré	Le processus continue

Appels système et réception de signaux

- Certains appels système dits "*lents*" peuvent être interrompus par des signaux ;
- Généralement, l'appel système retourne -1 et `errno` est renseignée : elle prend la valeur "`EINTR`".

Attention

Lorsqu'un processus est bloqué en attente d'une connexion (`socket`) ou de la fin d'un processus fils (`wait`), il faut vérifier la valeur de retour.

Table des matières

Signaux dits non-fiables

- Il est possible de modifier les actions associées à un signal (sauf pour certains comme `SIGKILL`) ;
- Pour modifier les actions associées à un signal :

```
void (*signal(int signum, void (*handler)(int)))(int)
```

- Les paramètres sont :
  - `signum` : le numéro du signal ;
  - `handler` est le gestionnaire à appliquer :
    - `SIG_IGN` : le signal est ignoré ;
    - `SIG_DFL` : le comportement par défaut est replacé ;
    - Une fonction `void handler(int signum)` : cette fonction est appelée lorsque le signal est reçu.
- Lorsqu'une fonction est appelée comme gestionnaire et qu'un signal est reçu :
  1. Le gestionnaire est reconfiguré à SIG_DFL ou le signal est bloqué ;
  2. La fonction "`handler`" est appelée.

Exemple

```
#include <stdio.h>  // Pour printf
#include <signal.h> // Pour la fonction signal

void handler(int signum) {
  // Le signal SIGINT a été reçu
  printf("Le signal SIGINT a été reçu\n");
}

int main(int argc, char *argv[]) {
  // La fonction handler est placée comme gestionnaire
  //  pour le signal SIGINT
  signal(SIGINT, handler);

  // Boucle infine
  while(1);

  return 1;
}
```

Signaux non-fiables ?

- Dans la version originale d'Unix, la fonction "`signal`" réinitialisait le gestionnaire à `SIG_DFL` après chaque appel ;
- Un gestionnaire pour un signal `SIGINT` devient :

```
void handler(int signum) {
  // Repositionnement du gestionnaire
  signal(signum, handler);

  // Traitement proprement dit
  printf("Le signal SIGINT a été reçu\n");
}
```

- Problème : entre l'appel et le repositionnement, un signal peut être perdu ;
- Dans BSD, le gestionnaire n'est pas replacé mais les nouvelles occurrences sont bloquées ;
- Comportement adopté dans la dernière version de la bibliothèque standard du C (lib6).

Les ensembles de signaux

- Les signaux sont gérés via des ensembles de signaux : `sigset_t`
- Ces ensembles sont manipulés via les fonctions suivantes :
  - `int sigemptyset(sigset_t *set)` :
    $\Longrightarrow$ vide l'ensemble (tous les signaux sont désélectionnés) ;
  - `int sigfillset (sigset_t *set)` :
    $\Longrightarrow$ remplit l'ensemble (tous les signaux sont sélectionnés) ;
  - `int sigaddset(sigset_t *set, int signum)` :
    $\Longrightarrow$ ajoute le signal "`signum`" à l'ensemble "`set`" ;
  - `int sigdelset(sigset_t *set, int signum)` :
    $\Longrightarrow$ supprime le signal "`signum`" de l'ensemble "`set`" ;
  - `int sigismember(const sigset_t *set, int signum)` :
    $\Longrightarrow$ indique si le signal "`signum`" appartient à l'ensemble "`set`" (1 si oui, 0 sinon).
- Ces fonctions (sauf `sigismember`) retournent 0 en cas de réussite ou -1 en cas d'erreur (erreur `EINVAL` si le signal n'est pas valide).

Modification des actions associées à un signal (1/2)

- La fonction pour modifier les actions des signaux :

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

- La structure "sigaction" est la suivante (peut varier suivant le système) :

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

  - sa_mask : ensemble de signaux à bloquer pendant l'exécution du gestionnaire ;
  - sa_flags : ensemble d'attributs modifiant le comportement du gestionnaire.

Modification des actions associées à un signal (2/2)

- Si l'attribut "sa_flags" contient l'indicateur "SA_SIGINFO", le gestionnaire est :

```
void (* sa_sigaction) (int, siginfo_t *, void *)
```

- La structure "siginfo_t" contient des attributs permettant de déterminer entre autres :
  - si_signo : le numéro du signal ;
  - si_errno : le numéro d'erreur ;
  - si_pid : le PID du processus émetteur du signal (si le signal est SIGCHLD) ;
  - ...
- Si l'indicateur "SA_SIGINFO" n'est pas spécifié :

```
void (* sa_handler)(int)
```

Bloquer des signaux

- La fonction est la suivante :

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset)
```

$\Longrightarrow$ Où :

- `how` peut prendre 3 valeurs :
  - `SIG_BLOCK` : les signaux de l'ensemble "`set`" sont ajoutés à l'ensemble des signaux déjà bloqués ;
  - `SIG_UNBLOCK` : les signaux de l'ensemble "`set`" sont supprimés de l'ensemble des signaux bloqués ;
  - `SIG_SET` : les signaux bloqués sont exactement ceux de l'ensemble "`set`".
- `set` : un ensemble de signaux ;
- `oldset` : s'il n'est pas nul, la valeur précédente du masque des signaux est placée dans "`oldset`".

Les fonctions "sigpending" et "sigsuspend"

- Pour récupérer l'ensemble des signaux qui se sont déclenchés pendant qu'ils étaient bloqués :

```
int sigpending(sigset_t *set)
```

$\Longrightarrow$ L'ensemble "set" reçoit les signaux qui se sont déclenchés.

- Remplacer temporairement l'ensemble des signaux bloqués :

```
int sigsuspend(const sigset_t *mask)
```

$\Longrightarrow$ L'ensemble des signaux bloqués est maintenant "mask" et le processus courant est endormi jusqu'à l'arrivée d'un signal.

Exemple (sans gestion d'erreur)

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

int main() {
  sigset_t sigs_new;                                  // Signaux à bloquer
  sigset_t sigs_old;                                  // Ancien masque

  sigfillset(&sigs_new);                              // Tous les signaux
  sigdelset(&sigs_new, SIGINT);                       // Sauf SIGINT
  sigdelset(&sigs_new, SIGQUIT);                      // Sauf SIGQUIT
  sigprocmask(SIG_BLOCK, &sigs_new, &sigs_old);       // Bloque les signaux

  /* Action A : tous les signaux sont bloqués sauf SIGINT et SIGQUIT */

  sigprocmask(SIG_SETMASK, &sigs_old, 0);             // Replacer par défaut
  sigemptyset(&sigs_new);                             // Aucun signal
  sigaddset(&sigs_new, SIGINT);                       // Plus SIGINT
  sigaddset(&sigs_new, SIGQUIT);                      // Plus SIGQUIT
  sigprocmask(SIG_BLOCK, &sigs_new, &sigs_old);       // Bloque les signaux

  /* Action B : seuls les signaux SIGINT et SIGQUIT sont bloqués */

  sigprocmask(SIG_SETMASK, &sigs_old, 0);             // Replacer par défaut

  return 1;
}
```

Table des matières

Exercice : un programme simple

Enoncé

1. Écrire un programme qui s'arrête après 3 réceptions du signal `SIGINT` ;
2. Décrivez la procédure pour exécuter ce programme et lui envoyer les signaux `SIGINT` ;
3. Peut-on écrire un programme qui ignore `SIGKILL` ? Pourquoi ?

Exercice : les signaux de fumée (sans fumée)

Enoncé

Nous désirons écrire deux programmes : le premier envoie des messages en morse au second qui doit les traduire et les afficher à l'écran. Pour cela, nous proposons d'utiliser les signaux. Nous nous contenterons d'envoyer des chiffres dont les codes en morse sont les suivants :

1	.----	3	...--	5	.....	7	--...	9	----.
2	..---	4	....-	6	-....	8	---..	0	-----

1. Proposez un protocole de communication en utilisant les signaux entre les deux programmes ;
2. Écrire un programme qui prend en paramètre le PID du programme traducteur et qui lui transfert un entier ;
3. Écrire le traducteur.