

# Architecture et Systèmes

Stefan Schwoon

Cours L3, 2019/20, ENS Cachan

# Signaux

---

Les **signaux** fournissent un interface primitif pour l'interaction entre les processus.

Un signal est un message envoyé à un processus soit par le noyau, soit par un autre processus (en passant par un appel système).

⇒ utilisé pour gérer des tâches de bas niveau, p.ex. signaler que des données sont disponibles, reveiller ou terminer un processus;

⇒ pas vraiment conçu pour l'échange des données.

# C'est quoi un signal ?

---

Un signal est un message simple. POSIX en définit une trentaine, les systèmes d'exploitation en utilisent souvent plus (pour l'usage interne).

Liste des signaux : `kill -l` sur la ligne de commande

Il est préférable d'utiliser les noms plutôt que les valeurs numériques (p.ex. pour compatibilité).

---

Quelques exemples:

**SIGINT** – généré par Ctrl+C dans la console

**SIGTERM** – pour terminer un processus

**SIGKILL** – tuer un processus (ne peut être ignoré/annulé)

**SIGUSR1** – usage libre

**SIGTSTP** – généré par Ctrl+Z dans la console.

**SIGALRM** – utilisé par sleep(3), alarm(2)

On note qu'il existent plusieurs signaux pour terminer des processus, à utiliser dans des circonstances différentes.

# Transfert d'un signal

---

Le transfert d'un signal comprend deux étapes :

**Envoi:** Le signal est enregistré pour processus *A* (le signal est dit *en attente*).

**Livraison:** Le système fait réagir processus *A* à un signal en attente.

# Envoi

---

Appel système: `kill`

p.ex., `kill(1000, SIGTERM)` envoie le signal `TERM` au processus 1000.

→ Le système stocke l'information que processus 1000 possède un signal de type `TERM` en attente.

Restrictions:

Un processus envoyant un signal doit appartenir soit à l'administrateur (`root`) soit au même utilisateur que le processus ciblé.

Pour chaque type de signal, un processus possède au plus un seul signal en attente à un moment donné. P.ex., si le signal `TERM` est en attente pour processus  $P$ , un deuxième envoi de `TERM` à  $P$  ne change rien.

---

D'autres moyens pour envoyer un signal (qui reviennent à l'appel `kill`) :

`kill` sur la ligne de commande

Dans un terminal, certaines combinaisons de clavier (Ctrl+C, +Z, +S, +Q, ...) font que le terminal envoie un signal à son "processus actuel".

Certaines fonctions arrangent pour le système d'envoyer un signal après un délai (`sleep`, `alarm`).

# Livraison

---

Quand l'ordonnanceur décide de donner un créneau de calcul à un processus, il vérifie d'abord l'existence d'un signal en attente.

Si c'est le cas, on procède avec la **livraison** du signal.  
L'exécution normale du processus continue après cette livraison.

Pour chaque type de signal, le processus possède une **disposition** actuelle.

Dispositions possibles : **Ign** (ignorer le signal), **Term** (terminer le processus), **Core** (terminer et créer un fichier *core*), **Stop/Cont** (suspendre/continuer le processus), ou un **gestionnaire de signal** individuel.

Voir aussi `signal(7)`.

# Modifier la disposition d'un signal

---

Un processus peut modifier sa disposition pour un signal avec `signal(2)` (déconseillé) ou `sigaction(2)`.

Pour certains signaux, la disposition ne peut être modifiée (notamment SIGKILL).

Un *gestionnaire* individuel est un pointeur vers une fonction qui sera exécutée lors d'une livraison.

# Attention

---

La livraison d'un signal peut interrompre certains fonctions avant leur terminaison normale.

Exemples : `sleep`, `wait`

→ tester le code renvoyé par ces fonctions, notamment quand on joue avec des signaux !

Note : Ces fonctions ne seont as interrompues par les signaux dont la disposition est `lgn`.

# Hérité des dispositions

---

Lors d'un `fork`, le fils hérite les dispositions de son père.

Cependant, `exec` remet les dispositions pour tout gestionnaire individuel à la disposition standard pour ce signal.

# Signaux diverses

---

**SIGCHLD** : Envoyé lorsqu'un processus fils termine

(aussi lorsqu'il est stoppé dans certains cas, voir ci-dessus).

Peut servir comme alternative pour `wait`, lire attentivement la documentation de sigaction !

**SIGALRM** : Utilisé par `alarm` (et, selon la réalisation du système) par `sleep`.

# Stop/Continue

---

**SIGSTOP** / **SIGCONT** : arrêter (temporairement) et continuer l'exécution d'un processus. La disposition de SIGSTOP ne peut pas être modifiée.

**SIGTSTP** : comme SIGSTOP, mais peut être ignoré. Envoyé par Ctrl+S dans la console (Ctrl+Q pour SIGCONT).

# Groupes de processus

---

Chaque processus appartient à un **groupe de processus**.

Les groupes possèdent un identifiant numérique.  
(typiquement identique au PID d'un membre)

`setpgid` peut changer le groupe d'un processus. Exemples :

`setpgid(p, g)` – rajoute  $p$  au groupe  $g$

`setpgid(0, 0)` – équivalent à `setpgid(p, p)`, où  $p$  est le PID du processus

# Groupes de processus et signaux

---

Un signal peut être envoyé à tous les membres d'un groupe.

P.ex., `kill` interprète un argument négative comme le numéro d'un groupe. (voir aussi `killpg`)

Exaeples: `kill(SIGINT, -100)` – envoyer SIGINT au groupe 100

# Groupes dans le shell

---

Un shell est exécuté dans un **terminal**.

Le terminal possède la notion d'un *groupe de premier plan*.

Les entrées de l'utilisateur (et les signaux en raison de Ctrl+C etc) sont envoyés par le terminal à ce groupe.

Quand le shell lance un commande, il crée un nouveau groupe pour ce processus et en fait le group de premier plan.

# Foreground/background jobs

---

**Ctrl+Z** dans le terminal envoie SIGTSTP au groupe *de premier plan*.

Comportement par défaut : Les processus s'arrêtent, le shell redevient le *processus de premier plan*.

**bg** et **fg** envoient SIGCONT à ce groupe ce qui leur permettra de continuer en premier plan ou en arrière plan.