

# Architecture et Systèmes

Stefan Schwoon

Cours L3, 2016/17, ENS Cachan

# Signaux

---

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

Un signal est un message envoyé a un processus soit par le noyau, soit par un autre processus.

⇒ 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 définit une trentaine de signaux, 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)

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 généré pour processus *A* par le noyau ou un autre processus (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 avec id 1000.

→ Le système mémorise que processus *A* possède un signal de type `TERM` en attente.

Restrictions:

Le processus *A* doit appartenir au même utilisateur que le processus qui envoie le signal (ou ce dernier doit être privilégié, p.ex. le noyau).

Un processus possède un seul signal d'un même type en attente au même temps. Si un autre signal du même type est envoyé avant la livraison, cet envoi est ignoré.

---

Autres moyens pour envoyer un signal (qui reviennent à `kill`) :

`kill` sur la ligne de commande

certaines combinaisons dans la console (Ctrl+C, +Z, +S, +Q, ...) – c'est le terminal qui envoie ces signaux

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

# Livraison

---

Quand processus *A* est choisi par l'ordonnanceur, celui-ci 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, *A* possède une **disposition** actuelle.

Dispositions possibles : **Ign** (ignorer le signal), **Term** (terminer le processus), **Core** (terminer et créer un fichier *core*), **Stop/Cont** (stopper/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 dispositions ne peut être modifiée (notamment SIGKILL).

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

# Attention

---

Les livraisons d'un signal peuvent interrompre certaines fonctions avant leur terminaison normale.

Exemples : `sleep`, `wait`

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

Note (1): Un signal ignoré ne compte pas comme “délivré” et ne peut interrompre ces fonctions.

# signal vs sigaction

---

L'usage de `signal` n'est plus encouragé, sauf pour mettre une disposition à `SIG_DFL` ou `SIG_IGN`.

`sigaction` est plus compliqué à utiliser, mais portable et offre plus d'options pour gérer certains détails

P.ex., le drapeau `SA_RESTART` gère si `wait` et certains autres opérations sont terminées par la livraison d'un signal.

# Héritage des dispositions

---

Lors d'un `fork`, le fils hérite des 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).

# Réentrance

---

Attention, l'utilisation des gestionnaires peut avoir des conséquences inattendues.

Notamment, la livraison d'un signal peut intervenir lorsque le processus est en train d'exécuter une fonction de librairie.

Si le gestionnaire fait appel à cette même fonction, celle-ci doit être *réentrante* ! (c.à.d. qu'elle doit pouvoir gérer ce genre de situation)

La plupart des fonctions de librairie sont réentrantes, mais pas p.ex. `malloc` !

Attention donc avec les opérations compliqués dans les gestionnaires, faire plutôt des choses simples.

# 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.

# Blockage des signaux

---

Un processus peut retarder la livraison d'un signal à l'aide de son **masque de blockage**

⇒ c'est pour bloquer *temporairement*, sinon utiliser SIG\_IGN.

Ce masque peut être altéré par **sigprocmask(2)**.

Un signal dans le masque reste en attente et sera livré lorsque son type est enlevé du masque.

En plus, des signaux peuvent être bloqués pendant une livraison, voir **sigaction**.

**sigsuspend(2)** permet de remplacer le masque en attendant un signal, puis restaurer (atomiquement) le masque.