# Langage C et Programmation Système
# TP n° 12 : Implémentation d'un shell – Partie 3
## *Signaux et jobs*

**Suite du TP n° 11 :** Le but de ce TP est de finaliser notre shell simplifié avec l'outil interne d'envoi de signaux `kill`, la gestion des signaux générés à travers le clavier, et le contrôle de jobs. Il vous est donc demandé de repartir du travail réalisé dans les TPs précédents. (Vous pouvez aussi vous servir du corrigé disponible sur DidEL.)

### Exercice 1 : La commande `kill` — *Envoyer des signaux*

Le but de cet exercice est de réaliser la commande spécifié ci-dessous. Les questions qui vous conduiront pendant l'implémentation suivent cette description.
*Synopsis :* `kill [-<sigspec>] [<pid>|<jobspec>]`
*Description :* `kill` envoie le signal nommé par `sigspec` aux processus nommés par `<pid>` ou `<jobspec>`. L'option `<sigspec>` est soit un nom de signal (comme `SIGKILL`), soit un numéro de signal. Si `<sigspec>` n'est pas présente, on suppose que le signal est `SIGTERM`, c'est à dire 15. L'argument `<pid>` est un identifiant de processus (PID), l'argument `<jobspec>` est une chaine de caractères de la forme `%n`, où `n` est un numéro (entier positif) identifiant un job. La commande `kill` renvoie 0 si au moins un signal a été envoyé avec succès, et 1 si une erreur s'est produite ou si une option illégale a été rencontrée.

1. Rajoutez une première version de la commande interne `kill` (qui masquera la commande externe du même nom). On supposera que l'option `<sigspec>` n'est pas présente et que l'argument est nécessairement un PID. (Indice : `kill(2)`.)
2. Implémentez la gestion de l'option `<sigspec>`, en utilisant les numéros et les noms de signaux décrits dans `signal(7)`.
3. Implémentez une version primitive de la gestion des arguments de type `<jobspec>`, en supposant que chaque job contient un seul processus, alors que ce n'est pas nécessairement le cas (un pipeline en arrière-plan peut en avoir plusieurs).

### Exercice 2 : Gestion de signaux — *Attraper des signaux*

Certains signaux sont dédiés à l'exécution interactive de processus et à la gestion de jobs. Dans cet exercice, on rendra le shell conscient de ces signaux, et aussi résistant à une fermeture non désirée.

1. Testez le comportement du shell face au raccourci clavier `Ctrl+C`, que le terminal virtuel traduit normalement par le signal `SIGINT`.
2. Écrivez dans le fichier `main.c` la procédure `static void sigs_catcher(int signum)`, qui s'occupe de gérer certains signaux. Si l'argument `signum` est égal à `SIGINT`, `SIGQUIT`, `SIGTSTP`, `SIGTTIN`, `SIGTTOU`, ou `SIGCHLD`, elle affiche simplement sur le flux d'erreur standard le message «`mysh: Recu signal <signum>.`».
3. Écrivez dans `main.c` la fonction `sigs_setup()`, qui définit la fonction `sigs_catcher` comme attrapeuse des signaux de la question précédente. Rajoutez dans la fonction `main`, avant la boucle principale, l'appel à `setup_sigs()`. Indice : `signal(2)`.

**4.** Re-testez le comportement du shell :

1. Que se passe-t-il lorsqu'une commande externe termine son exécution ?
2. Et si la commande est interne ?
3. Que se passe-t-il quand on saisit `Ctrl+C` sur le clavier ? Et avec `Ctrl+Z` ?

**Exercice 3 : Gestion de jobs** ***Signaux et groupes de processus***

Dans cet exercice, on va implémenter une gestion de jobs primitive. Pour tester le shell, on l'exécutera directement dans un terminal virtuel, comme `gnome-terminal`, `konsole` ou `xterm`, c'est-à-dire en dehors d'une session `bash` interactive, et sans aucun wrapper, comme ceux de readline (`ledit` ou `rlwrap`). Indice : l'option `-e` des terminaux virtuels permet de spécifier la commande à exécuter.

**1.** D'abord, on généralise la création et la destruction de jobs, même pour les commandes en avant-plan. Rajoutez à `jobs.c` deux variables entières globales : `jobcur`, qui contiendra l'identifiant du job courant (`-1` indiquera son absence), et `jobbackground`, qui sera vrai si et seulement si le job courant est en arrière-plan. Modifiez la fonction exécutant des commandes externes et ses variantes avec redirections ou pipeline, pour qu'elles créent un nouveau job, et pour qu'elles le détruisent à la fin de l'exécution, toujours en tenant à jour les différentes variables d'état.

**2.** On implémente la gestion de `Ctrl+Z`. Changez la gestion de `SIGTSTP` : si on reçoit ce signal et que le job courant est en avant-plan, alors on lui émet un `SIGSTOP`. Modifiez les fonctions d'exécution externe : l'attente du processus principal du shell peut maintenant se terminer dans le cas de stoppage. Dans ce cas, on retourne `0`.

**3.** On implémente la gestion de `Ctrl+C`. Changez la gestion de `SIGINT` : si on le reçoit et que le job courant est en avant-plan, on lui émet un `SIGINT`.

**4.** Rajoutez la commande interne `bg` qui prend un identifiant de job selon la syntaxe de `jobspec` et signale au job de continuer son éxecution, mais n'attend pas sa terminaison. Pour ce faire, elle émet un `SIGCONT` au processus principal du job.

**5.** Les raccourcis clavier et les commandes `kill`, `bg` opèrent seulement sur le PID principal d'un job. On rajoute maintentant le support des groupes de processus POSIX : un identifiant unique pour chaque job.

1. Modifiez les fonctions d'exécution externe pour que le processus enfant crée un nouveau groupe avec son propre PID. Indice `setpgid(2)`.
2. Modifiez la gestion de `SIGINT` et `SIGTSTP`, et les commandes `kill`, `bg` pour que les signaux soient émis au groupe de processus. Indice `kill(2)` sur un PID négatif.

**6.** (Optionnel.) Améliorez `kill` : le nouveau paramètre `<pidspec>` de `kill` permet d'utiliser les séquences spéciales ci-dessous :

- `%+`, `%%`, et `%` indiquent le job courant, c'est-à-dire le dernier job qui a été stoppé tandis qu'il était au premier plan, ou qui a été lancé en arrière-plan.
- `%-` est le job précédent. S'il n'y a qu'un seul job, `%-` est équivalent à `%+`

**7.** (Optionnel.) Rendez l'affichage des jobs plus agréable pour l'utilisateur : définissez une structure `job_t` contenant non seulement le PID du processus principal, mais aussi son nom (l'argument `argv[0]` de la commande executée). N'oubliez pas d'adapter les fonctions de création et de destruction.