Rapport SYSG5 : Contrôle de Jobs

Filipe PEREIRA MARTINS - 58093

# **Introduction**

Le contrôle des jobs dans un système d'exploitation Linux est une fonctionnalité essentielle qui permet aux utilisateurs du *shell* d'exécuter simultanément plusieurs commandes, tout en maintenant un contrôle précis sur leur exécution. Cette fonctionnalité permet d'assigner un job en "*foreground*", tandis que d'autres s'exécutent en "*background*", offrant ainsi une souplesse remarquable dans la gestion des tâches. Elle permet de stopper, de déplacer, et de surveiller les jobs en cours, offrant ainsi une solution pratique pour effectuer plusieurs opérations sans la nécessité d'ouvrir de nouvelles terminales.

Cette recherche a été entreprise dans le but d'explorer en profondeur les mécanismes qui sous-tendent le contrôle des jobs dans Linux. Le rapport se compose de trois sections principales, chacune se focalisant sur un aspect spécifique du contrôle des jobs : les groupes de processus, les sessions de processus et le terminal de contrôle. En combinant ces trois sections, l’objectif est de fournir une vue d'ensemble complète du contrôle des jobs dans Linux, en commençant par ses bases et en progressant vers des techniques plus avancées.

# **Groupes de Processus**

## **Définition**

Un groupe de processus est une collection d’un ou plusieurs processus qui partagent un identifiant de groupe de processus (*PGID*) commun. L'identifiant de groupe de processus (*PGID*) est un numéro de type *pid\_t*, similaire à l'identifiant de processus (*PID*). Chaque groupe de processus est dirigé par un chef de groupe de processus, qui est le processus responsable de la création du groupe et dont l'identifiant de processus (*PID*) devient l'identifiant de groupe de processus (*PGID*) du groupe. Lorsqu'un nouveau processus est créé, il hérite automatiquement de l'identifiant de groupe de processus (*PGID*) de son processus parent.

## **Création d’un nouveau groupe**

### **L’appel système *setpgid()***

L’appel système *setpgid (pid\_t pid, pid\_t pgid)* permet de modifier le groupe de processus auquel appartient un processus donné ou créer un nouveau groupe de processus.

Paramètres :

* *pid\_t pid :* C’est le PID du processus dont le groupe de processus doit être modifié (PGID). Si sa valeur est 0 alors la valeur du paramètre sera la même que le PID du processus appelant.
* *pid\_t pgid :* C’est le PGID vers lequel le processus avec le PID spécifié doit être déplacé. Si le paramètre est spécifié comme 0, alors sa valeur sera la même qui le PID du processus appelant.

Si les arguments *pid* et *pgid* spécifient le même processus (c'est-à-dire que *pgid* vaut 0 ou correspond au PID du processus spécifié par *pid*), un nouveau groupe de processus est créé et le leader de ce nouveau groupe est le processus spécifié (processus appelant si pid vaut 0).

Une restriction importante à savoir est qu’un processus ne peut pas modifier le PGID d’un de ses enfants lorsque celui-ci a effectué un *exec ().* Si ça arrive une erreur *EACCES* sera déclenché.

### **L’utilisation dans le contrôle de jobs**

Dans le contexte du contrôle de jobs, il est essentiel de regrouper tous les processus d'un job (par exemple, une commande ou un pipeline) dans un seul groupe de processus. Cela permet au *shell* de gérer efficacement ces processus en envoyant simultanément des signaux à tous les membres du groupe de processus. Cependant, une problématique se pose : comment modifier le PGID (des processus enfants d'un des processus d'un job ? Cette question découle de l'imprévisibilité de l'ordonnanceur lors de l'appel à *fork()* : nous ne pouvons pas garantir quel processus, le père ou le fils, sera le premier à s'exécuter.

Pour résoudre ce problème, les *shells* de contrôle de jobs sont programmés de manière à ce que le père et le fils appellent tous les deux *setpgid()* immédiatement après l'appel à *fork().* Cette approche garantit que le PGID du fils est modifié, indépendamment de l'ordre d'exécution des processus. Si le père n’est pas le premier à appeler *setpgid(),* il recevra une erreur EACCES puisque le fils a déjà changé son PGID et a effectué un exec(), pour remédier à ce problème il va tout simplement ignorer l’occurrence de cette erreur après l’appel de *setpgid().*

# **Sessions**

## **Définition**

Une session est un regroupement de plusieurs groupes de processus. La participation d'un processus à une session est déterminée par son identifiant de session (SID). Le leader de session est le processus responsable de la création d'une nouvelle session et dont sont identifiant (PID) devient l'identifiant de session (SID) de cette session. Lorsqu'un nouveau processus est créé, il hérite automatiquement de l'identifiant de session (SID) de son processus parent.

## **Création d’une session**

### **L’appel système *setsid()***

L’appel système pid\_t setsid(void) permet de créer une nouvelle session et définir le processus appelant comme le leader de cette session. Le PGID et SID deviennent le même que le PID du processus appelant.

Il est important de noter que, d'une part, un leader de groupe de processus existant ne peut pas créer une nouvelle session. D'autre part, lorsqu'un processus fait l'appel à *setsid()*, il perd toute connexion préexistante au terminal de contrôle.

# **Terminales de contrôle**

## **Association** **à un terminal de contrôle**

## **Dissociation à un terminal de contrôle**

## **Processus en *Background* et *Foreground***

### **L’accès au terminal (SIGTTIN et SIGTTOU)**

## **Le signal SIGHUP**

# **Contrôle de jobs**

## **L’utilisation dans le *shell***