Rapport SYSG5 : Controle de jobs

Filipe PEREIRA MARTINS - 58093

Novembre 2023

Table des matières

1 Introduction **2**

2 Groupes de Processus **3**
2.1 Création d'un nouveau groupe 3
2.1.1 L'appel système *setpgid()* 3
2.1.2 L'utilisation dans le contrôle de jobs 4

3 Sessions **5**
3.1 Création d'une session 5
3.1.1 L'appel système *setsid()* 5
3.2 Disposition des processus en groupes/sessions 5

4 Terminales de contrôle **6**
4.1 Association à un terminal de contrôle 6
4.2 Le signal SIGHUP 6
4.2.1 *nohup* 7

5 Références **8**

1 Introduction

Le contrôle des jobs dans un système d'exploitation Linux permet l'exécution simultanée de plusieurs commandes avec un contrôle précis. Cette fonctionnalité permet d'assigner un job en "foreground" pendant que d'autres s'exécutent en "background", simplifiant ainsi la gestion des tâches. On peut arrêter, déplacer et surveiller les jobs en cours, offrant une solution pratique pour effectuer diverses opérations sans ouvrir de nouveaux terminaux.
Cette recherche a été initiée pour approfondir la compréhension des mécanismes régissant le contrôle des jobs dans l'environnement Linux. Le rapport est structuré en plusieurs sections, chacune se concentrant sur un aspect particulier du contrôle des jobs. L'objectif principal est de présenter une vue d'ensemble exhaustive de cette fonctionnalité, en débutant par ses fondements pour ensuite explorer des concepts plus avancés.

2 Groupes de Processus

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.

2.1 Création d'un nouveau groupe

La création d'un nouveau groupe de processus, fondamentale dans le contrôle des jobs, est réalisée via l'appel système *setpgid()* . Cette fonction permet de changer le groupe de processus d'un processus existant ou de créer un groupe totalement nouveau. Nous examinons ici cet appel système ansi que les conséquences de l'utilisation de *setpgid()*.

2.1.1 L'appel système *setpgid()*

```
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
```

L'appel système *setpgid()* 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.

2.1.2 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 (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 ? Comme expliqué précédemment, tous les processus d'un job doivent être dans un seul groupe, ce qui veut dire que tout processus fils créé par ce job doit lui aussi être dans le même groupe que son parent.
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()*.

3 Sessions

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 (généralement le shell). Lorsqu'un nouveau processus est créé, il hérite automatiquement de l'identifiant de session (SID) de son processus parent.

3.1 Création d'une session

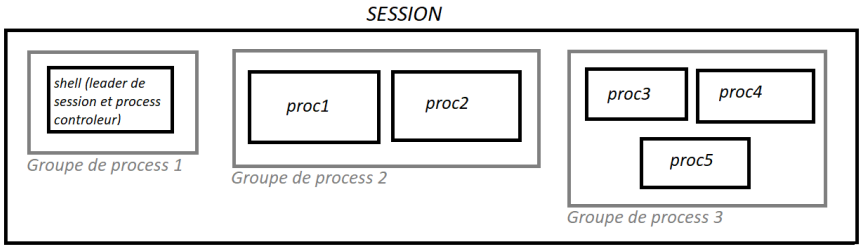
3.1.1 L'appel système *setsid()*

```
#include <unistd.h>

pid_t setsid(void);
```

L'appel système *setsid()* 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.

3.2 Disposition des processus en groupes/sessions



4 Terminales de contrôle

Un terminal de contrôle est un terminal spécial associé à une session de processus. Il sert de point de communication principal entre une session et l'utilisateur, permettant aux processus de cette session d'interagir avec l'utilisateur via le terminal. Tous les processus d'une session peuvent avoir un seul et unique terminal de contrôle.

4.1 Association à un terminal de contrôle

Comme expliqué précedemment, lorsqu'une nouvelle session est créée, le processus leader de cette session n'a pas encore de connexion au terminal de contrôle. En d'autres termes, à sa création, une session ne dispose pas encore d'un terminal de contrôle associé.
Pour qu'une session puisse établir une connexion avec un terminal de contrôle, le processus leader de cette session doit ouvrir un nouveau terminal de contrôle. Il est essentiel de noter que ce terminal ne doit en aucun cas être déjà associé à une autre session.
Lorsque le leader de la session réussit à établir cette connexion avec le terminal, il devient ce que l'on appelle le "processus contrôleur". Les détails sur les implications de ce rôle seront expliqués dans la suite.
Tous les processus dans une session héritent de l'accès du terminal de contrôle. Habituellement, les nouvelles sessions sont créées par le programme de login du système et le leader de la session est le processus exécutant le shell de login de l'utilisateur.

4.2 Le signal SIGHUP

Lorsqu'une session (c'est-à-dire le processus contrôleur) perd son terminal de contrôle ou le processus contrôleur se termine (fermeture de la fenêtre du terminal par exemple), tous les processus de la session perdent également leur association à ce terminal de contrôle.
Pour les informer de cet évènement, le noyau envoie un signal (SIGHUP) au processus contrôleur, qui lui-même va créer une réaction en chaine en envoyant SIGHUP à tous les groupes de processus que lui-même a créé, à noter que ce signal ne sera pas envoyé à des groupes de processus (jobs) qui n'ont pas été créées par le processus contrôleur !
Le signal SIGHUP, par défaut termine un programme, mais celui-ci peut bien sur créer un handler, et continuer à s'exécuter (pour faire des opérations de nettoyage par exemple). Le signal SIGCONT peut éventuellement être envoyé s'il y avait des processus stoppés lorsque qu'il y a la dissociation du terminal de contrôle.

4.2.1 *nohup*

Une commande particulière qui permet d'ignorer le signal SIGHUP, est la commande *nohup*. Cette commande permet d'exécuter un processus/job et ignorer SIGHUP, même lorsque le processus de contrôle ou le terminal de contrôle se termine (comme expliqué précédemment ces deux évènements génèrent le signal SIGHUP). Tout output généré par le processus/job sera redirigé vers nohup.out. Et bien sûr si le processus/job essaye de lire depuis le terminal de contrôle alors que celui-ci n'existe plus, une erreur sera retourné.

5 Références

— The Design and Implementation of the FreeBSD Operating System, by Marshall Kirk McKusick, George V. Neville-Neil
— The Linux Programming Interface by Michael KerrisK
— Advanced Programming in the UNIX Environment : Second Edition by W. Richard Stevens, Stephen A. Rago
— The GNU C Library (glibc) manual