Chapitre 6
Communication interprocessus (IPC)

1. Principes de communication interprocessus

Nous avons étudié dans les chapitres précédents différents mécanismes permettant à des processus de communiquer entre eux, pour échanger des données et pour se synchroniser. Il s'agit des tubes et des fichiers projetés en mémoire, pour les données, des signaux et des verrous de fichiers, pour la synchronisation.

D'autres mécanismes plus élaborés ont été mis en place au cours de l'évolution des systèmes de type Unix : les segments de mémoire partagée, les files d'attente de messages et les sémaphores.

Remarque

Les sockets, mécanisme d'échange de données en local, mais surtout à travers un réseau, font l'objet du chapitre suivant.

1.1 IPC System V et IPC POSIX

D'abord implémentés dans le cadre d'Unix System V, le système Unix développé par ATT, ces mécanismes de communication interprocessus (*Inter Process Communication*) ont ensuite été revus et améliorés pour la normalisation POSIX. C'est la raison pour laquelle, bien que les deux ensembles, IPC System V et IPC POSIX, proposent les mêmes types de fonctionnalités, Linux, comme la plupart des systèmes de type Unix, supporte les deux.

On considère généralement que les mécanismes IPC POSIX sont plus simples à programmer et corrigent certains défauts de conception des IPC System V, apparus à l'usage. Ils ont également l'avantage d'être normalisés et compatibles avec le multithreading. Cependant, les mécanismes IPC System V étant apparus plus tôt sont devenus un standard de fait du monde Unix, repris dans les spécifications SUS, et utilisés par de très nombreuses applications.

Il est donc recommandé d'opter pour les IPC POSIX dans le développement de nouvelles applications, mais il est utile, voire nécessaire, de connaître également les IPC System V pour pouvoir maintenir l'existant. C'est la raison pour laquelle nous présenterons les deux ensembles dans ce chapitre.

1.2 Fonctions des différents mécanismes d'IPC

Les trois mécanismes évoqués, segments de mémoire partagée, files d'attente de messages et sémaphores ont des rôles différents et souvent complémentaires.

Les segments de mémoire partagée servent à échanger des données entre processus de manière simple et optimisée, les sémaphores permettent d'assurer la synchronisation des processus, les files d'attente de messages, elles, permettent d'échanger des données de façon ordonnée entre processus, combinant ainsi des fonctions de partage et de synchronisation.

1.2.1 Segment de mémoire partagée

Nous avons vu que le noyau Linux alloue à chaque processus un espace mémoire réservé, que lui seul peut utiliser. Cette protection indispensable des processus les uns par rapport aux autres présente l'inconvénient de rendre difficile le partage d'informations entre processus. C'est la raison pour laquelle Linux propose des méthodes de partage de certaines zones mémoire entre processus : les segments de mémoire partagée. Une fois créées par un appel système spécifique, ces zones mémoire sont accessibles, sous réserve des permissions d'accès, aux autres processus. Le partage des données qu'elles contiennent est alors immédiat et très performant par nature, puisque tout se passe directement en mémoire utilisateur.

L'inconvénient de cette mémoire partagée, c'est qu'elle est délicate à gérer en écriture. Elle nécessite souvent un mécanisme complémentaire pour assurer la synchronisation des accès aux mêmes données par plusieurs processus.

1.2.2 Files d'attente de messages

Ce mécanisme d'échange de données se distingue des tubes par le traitement des données sous forme de messages indépendants et non en flot d'octets. D'autre part, les messages sont gérés par une file d'attente et peuvent être "typés", ce qui permet de les sélectionner selon différents critères et de ne pas forcément les lire dans l'ordre d'arrivée.

1.2.3 Sémaphores

Les sémaphores permettent une synchronisation efficace des processus. Ils sont généralement utilisés en lien avec une ressource partagée, un segment de mémoire par exemple, pour déterminer qui peut accéder à la ressource, dans notre exemple aux données du segment de mémoire, en lecture ou en écriture.

Les processus qui souhaitent obtenir l'accès à des ressources partagées se mettent en attente d'un ou plusieurs sémaphores, gérés par le noyau. Le processus qui obtient le sémaphore peut utiliser les ressources, puis relâcher le sémaphore pour permettre à un autre processus d'y accéder à son tour.

Contrairement aux verrous sur fichiers qui peuvent être rendus contraignants, les sémaphores mettent en place une synchronisation de type **collaboratif**, supposant que tous les processus concernés respectent la règle pour accéder aux ressources partagées.

2. Les segments de mémoire partagée

Les segments de mémoire partagée sont le moyen le plus performant pour partager des données entre processus. Une fois créés, ils sont accessibles à tout processus, sous réserve du contrôle d'accès mis en place par le processus créateur du segment de mémoire partagée. Ils sont persistants au niveau noyau, ce qui signifie que leur durée de vie maximale est celle du noyau, ils sont automatiquement supprimés à l'arrêt du système.

Une fois qu'un processus s'est attaché au segment, il peut accéder à la zone de données correspondante, en lecture et/ou en écriture, pour y gérer des variables, comme si elles étaient dans son espace mémoire virtuel.

La zone mémoire partagée est projetée dans l'espace virtuel du processus. Par conséquent, toute modification effectuée par un processus est immédiatement visible des autres processus attachés au segment. C'est la raison pour laquelle, sauf cas particuliers, il faut mettre en place un mécanisme de synchronisation entre les processus pour éviter les problèmes d'accès concurrents (*race conditions*).

Il existe deux mécanismes de segments de mémoire partagée utilisables avec Linux, ceux d'origine Unix System V et intégrés dans les standards SUS, ceux plus récents mis en place dans le cadre des normes POSIX. Nous allons décrire successivement les deux.

2.1 Les segments de mémoire partagée System V

Les IPC System V, segments de mémoire partagée, files d'attente de messages et sémaphores, partagent une logique et des outils communs. Leur implémentation s'appuie sur des attributs et des méthodes cohérents, facilitant leur utilisation, en contrepartie de la complexité de certains d'entre eux.

Un segment de mémoire partagée System V est un objet géré par le noyau, créé à la demande d'un processus et soumis à des permissions d'accès définies par le processus créateur.

Un processus qui souhaite accéder à un segment de mémoire partagée doit s'y attacher, en lecture et/ou en écriture, en fournissant l'identifiant du segment de mémoire souhaité. Le noyau contrôle les droits du processus et autorise ou non l'accès. Si le processus est autorisé à s'attacher au segment, le noyau en garde la trace. Une fois attaché, le processus reçoit une adresse, dans son espace mémoire virtuel, lui permettant d'accéder à la zone mémoire partagée. Le processus peut donc gérer cette zone relativement à l'adresse fournie, comme il le ferait avec une zone mémoire de son espace virtuel.

Quand un processus n'a plus l'usage du segment de mémoire partagée, il peut s'en détacher. Quand le processus se termine, le noyau le détache automatiquement du segment de mémoire partagée.

Un processus privilégié ou associé à l'UID du créateur ou du propriétaire du segment de mémoire partagée peut demander sa suppression. Cependant, cette suppression ne sera effective que lorsque plus aucun processus ne sera attaché au segment de mémoire partagée.

Remarque

La première implémentation des IPC System V sur Linux a été faite via un module dynamique du noyau, gérant un unique appel système pour l'ensemble des mécanismes IPC, `ipc()`. Différentes fonctions enveloppes spécialisées ont été intégrées dans la bibliothèque du langage C, `glibc`, pour utiliser les différents objets. Elles doivent être utilisées plutôt que l'appel système proprement dit, pour des raisons de facilité et surtout de compatibilité. Dans la suite du chapitre, nous décrirons donc ces fonctions, comme s'il s'agissait de fonctions enveloppes d'appels système distincts.

2.1.1 Création d'un segment de mémoire partagée : shmget()

L'appel système `shmget()` permet de créer un segment de mémoire partagée, ou d'obtenir son identifiant à partir de sa clef, s'il existe.

Syntaxe

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int flags);
```

Arguments

`key`	Clef d'identification du segment de mémoire partagée
`size`	Taille minimum en octets du segment de mémoire partagée
`flags`	Options

Valeur retournée

`-1`	Erreur, code erreur positionné dans la variable `errno`
`!= -1`	Identifiant unique du segment de mémoire partagée

Description

Si la clef d'identification fournie n'est pas déjà associée à un segment de mémoire partagée existant, le noyau le crée, de la taille indiquée (arrondie en un multiple de la taille d'une page mémoire), si l'option `IPC_CREAT` est spécifiée.

Une autre possibilité pour créer un segment de mémoire partagée est de spécifier la valeur `IPC_PRIVATE` comme argument `key`, auquel cas il est inutile de positionner l'option `IPC_CREAT`, le noyau créera un segment de mémoire partagée en générant une clef unique, et retournera l'identifiant du segment.

Si la clef est associée à un segment existant, et que la taille mémoire indiquée est inférieure ou égale à celle du segment, l'appel système retourne l'identifiant unique du segment, sauf si les options `IPC_CREAT` et `IPC_EXCL` sont spécifiées, auquel cas l'appel système retourne `-1` et positionne l'erreur `EEXIST` dans `errno`.