Programmation système : processus

Juliusz Chroboczek

24 Novembre 2009

1 Notion de processus

Un *programme* est un fichier contenant une suite d'instructions exécutables et les données associées. Lorsque ce programme est *exécuté*, il est d'abord chargé en mémoire puis exécuté par le processeur.

Un *processus* est une instance d'un programme en train de s'exécuter. Informellement, un processus c'est une zone mémoire et un flot de contrôle. Plus précisément, un processus consiste en :

- une zone de mémoire contenant le code exécutable (le segment de *texte*), les variables globales (le segment de *données*), les variables locales et la chaîne d'activation des fonctions (la *pile*) et les données allouées dynamiquement (le *tas*) ;
- un *contexte*, c'est à dire l'ensemble des registres du processeur, notamment celui qui indique la prochaine instrucution à exécuter et celui qui indique le sommet de la pile ;
- une petite structure du noyau, contenant des informations supplémentaires à propos du processus, tels que son numéro et son propriétaire.

Virtualisation de la mémoire Les adresses de mémoire manipulées par le code utilisateur ne correspondent pas forcément à des adresses physiques : la même adresse mémoire dans deux processus peut correspondre à deux locations physiques distinctes, et une adresse peut même ne pas correspondre à de la mémoire du tout, mais être simulée par des données stockées sur disque. La mémoire est donc *virtualisée*, le système présentant à l'utilisateur une abstraction qui ressemble à de la mémoire physique, mais peut être implémentée de diverses manières.

1.1 L' ordonnanceur

Unix est un système à *temps partagé* : plusieurs processus peuvent être exécutables à un moment donné, le système donnant l'illusion d'une exécution simultanée en passant rapidement d'un processus à l'autre[1]. Il s'agit là d'un autre cas de *virtualisation*, cette fois-ci du processeur.

La partie du noyau qui choisit quel processus doit s'exécuter à un moment donné s'appelle *l'ordonnanceur (scheduler)*. L'ordonnanceur maintient les structures de données suivantes (Figure 1(a)) :

1 Dans toute cette partie, nous faisons l'hypothèse simplificatrice d'un système à un seul processeur.

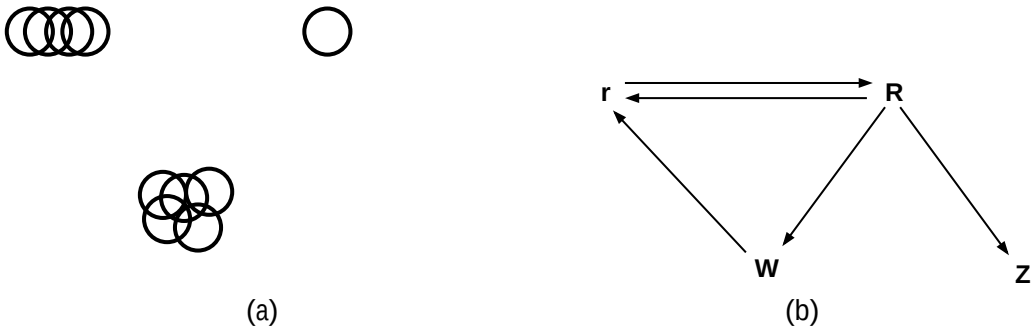

Figure 1: Structures de données de l'ordonnanceur (a) et transitions des processus (b)

– le processus en train de s'exécuter (R) ;
– une file de processus prêts à s'exécuter (r) ;
– un ensemble de processus en attente d'un événement [2] (W) ;
– un ensemble de processus morts, les *zombies* (Z).

Un processus peut donc avoir les états suivants (Figure 1(b)) :
– R, « *Running* » en train de s'exécuter ;
– r, « *runnable* », prêt à s'exécuter ;
– W, « *Waiting* », en attente d'un événement ;
– Z, « *Zombie* », mort.

Les transitions suivantes entre états sont possibles :
– $\varnothing \longrightarrow$ r, création d'un processus ;
– r $\longrightarrow$ R et R $\longrightarrow$ r, changement de contexte *(context switch)* décidé par l'ordonnanceur ;
– R $\longrightarrow$ W, appel système bloquant (voir ci-dessous) ;
– W $\longrightarrow$ r, arrivée d'un événement ;
– R $\longrightarrow$ Z, mort d'un processus.

1.2 Appels système bloquants

Lorsqu'un processus est dans l'état « en train de s'exécuter » R, il peut exécuter des instructions du processeur (par exemple pour effectuer des calculs), ou faire des appels système. Parfois, le noyau peut satisfaire l'appel système immédiatement — par exemple, un appel à `gettimeofday` calcule l'heure actuelle et retourne immédiatement au code utilisateur.

Souvent, cependant, un appel système demande une interaction prolongée avec le monde réel ; un processus exécutant un tel appel système est mis en attente d'un événement (état W), et ne sera réveillé (passé à l'état « prêt à s'exécuter » r) que lorsque l'appel système sera prêt à retourner. Un tel appel système est dit *bloquant*.

Considérons par exemple le cas d'un appel système `read` demandant des données stockées sur le disque. Au moment ou le processus effectue l'appel système, le contrôleur de disque est

2 En pratique, il s'agit d'un ensemble de files, une par évènement possible, ou même d'une structure plus complexe.

programmé pour lire les données[3], et le processus appelant est mis en attente de l'événement « requête de lecture terminée ». Quelques millisecondes plus tard, lorsque la requête aura abouti,

Un processus maintient deux propriétaires[4] : un *propriétaire réel*, qui représente l'utilisateur qui a créé ce processus, et un *propriétaire effectif*, qui code les privilèges dont dispose ce processus, par exemple pour modifier des fichiers.

Le propriétaire d'un processus peut être identifié à l'aide des appels système suivants :

```
uid_t getuid(void);
gid_t getgid(void);
uid_t geteuid(void);
gid_t getegid(void);
```

Les appels système `setuid`, `setgid`, `seteuid` et `setegid` permettent de changer de propriétaire. Leur sémantique est complexe, et ils ne sont pas portables : leur comportement diffère entre BSD et SYSV.

4 En fait, il y en a davantage — voyez `setresuid`(2) pour plus d'informations.