Programmation système : position courante, extension, troncation

Juliusz Chroboczek

28 Octobre 2009

1 Le pointeur de position courante

Les lectures et les écritures dans un fichier sont par défaut *séquentielles* : les données sont lues ou écrites les unes à la suite de l'autre. Par exemple, lors de la séquence de code

```
rc = write(fd, "a", 1);
rc = write(fd, "b", 1);
```

l'octet « b » est écrit après l'octet « a ».

La position dans un fichier à laquelle se fait la prochaine lecture ou écriture est stockée dans le noyau dans un champ de l'entrée de fichier ouvert qui s'appelle le *pointeur de position courante*. Lors de l'appel système open, le pointeur de position courante est initialisé à 0, soit le début du fichier.

1.1 L'appel système `lseek`

Le pointeur de position courante associé à un descripteur de fichier peut être lu et modifié à l'aide de l'appel système `lseek` :

```
off_t lseek(int fd, off_t offset, int whence)
```

Le paramètre `fd` est le descripteur de fichier dont l'entrée de la table de fichiers ouverts associée doit être modifiée. Le paramètre `offset` (« déplacement ») identifie la nouvelle position, et le paramètre `whence` (« à partir d'où ») spécifie l'interprétation de ce dernier. Il peut avoir les valeurs suivantes :

- `SEEK_SET` : `offset` spécifie un décalage à partir du début du fichier ;
- `SEEK_CUR` : `offset` spécifie un décalage à partir de la position courante ;
- `SEEK_END` : `offset` spécifie un décalage à partir de la fin du fichier.

L'appel `lseek` retourne la nouvelle valeur du pointeur de position courante, ou -1 en cas d'erreur (et alors `errno` est positionné).

> **Attention :** le type `off_t` est d'habitude un synonyme de `long`, et pas de `int` ; attention donc au type des variables.

Exemples Pour déplacer le pointeur de fichier au début d'un fichier, on peut faire

```
lrc = lseek(fd, 0L, SEEK_SET);
```

Pour déplacer le pointeur de fichier à la fin d'un fichier, on peut faire

```
size = lseek(fd, 0L, SEEK_END);
```

Si l'appel réussit, la variable `size` contient la taille du fichier.

On peut déterminer la position courante à l'aide de l'appel

```
position = lseek(fd, 0L, SEEK_CUR);
```

1.2 Le mode « ajout à la fin »

Il est très courant de vouloir ajouter des données à la fin d'un fichier. En première approximation, ce n'est pas difficile :

```
fd = open("/var/log/messages", O_WRONLY);
lrc = lseek(fd, 0L, SEEK_END);
rc = write(fd, ...);          /* condition critique ! */
close(fd);
```

Malheureusement, une telle approche mène à une condition critique *(race condition)* qui peut causer une perte de données. Considérons en effet se qui peut se passer si deux processus veulent simultanément ajouter des données à la fin du même fichier :

```
/* processus A */              /* processus B */
lseek(fd, 0L, SEEK_END);
                               lseek(fd, 0L, SEEK_END)
                               write(fd, "b", 1)
write(fd, "a", 1)
```

Supposons, pour fixer les idées, que le fichier a une taille de 1000 octets. Le processus A commence par se positionner à la fin du fichier, soit à la position 1000. Supposons maintenant qu'un changement de contexte intervient à ce moment, le contrôle passe au processus B, qui se positionne à la position 1000, et y écrit un octet « b ». Le contrôle repasse ensuite au processus A, qui est toujours positionné en 1000 ; il écrit donc un octet « a » à la position 1000, et écrase donc les données écrites par le processus B.

Une solution entièrement générale au problème de l'accès à des ressources partagées demande que les processus A et B effectuent une synchronisation, par exemple en posant des verrous sur la ressource commune ou en utilisant un troisième processus qui sert d'arbitre. Unix inclut cependant une solution simple pour le cas particulier de l'écriture à la fin du fichier.

Lorsque le drapeau `O_APPEND` est positionné dans le deuxième paramètre de `open`, le fichier est ouvert en mode *écriture à la fin (append mode)*. Le noyau repositionne alors le pointeur de position courante à la fin du fichier lors de toute opération d'écriture effectuée à travers ce pointeur de fichier. L'opération ayant entièrement lieu dans le noyau, celui-ci s'assure de l'exécution atomique du positionnement et de l'écriture.

2 Troncation et extension des chiers

Lorsqu'un processus écrit à la fin du fichier, la taille de ce fichier augmente ; on dit alors que le fichier est *étendu*. L'opération inverse à l'extension s'appelle la *troncation*.

2.1 Extension implicite

Toute écriture au delà de la fin d'un fichier cause une extension de celui-ci. Par exemple, la séquence de code suivante provoque une extension d'un octet :

```
lrc = lseek(fd, 0L, SEEK_END);
rc = write(fd, "a", 1);
```

Une extension plus importante peut être réalisée en se positionnant au-delà de la fin du fichier et en écrivant des données ; les parties du fichier qui n'ont jamais été écrites sont alors automatiquement remplies de 0 par le noyau. Par exemple, la séquence suivante ajoute à la fin du fichier 1 Mo de zéros suivi d'un octet « a » :

```
lrc = lseek(fd, 1024 * 1024L, SEEK_END);
rc = write(fd, "a", 1);
```

Parenthèse : chiers à trous En fait, les données non écrites ne sont pas forcément stockées sur disque : le noyau note simplement que la plage est « pleine de zéros », et produira des zéros lors d'une prochaine lecture. Ceci peut être constaté en consultant le champ `st_blocks` de l'i-nœud, par exemple à l'aide de « `ls -s` ».

2.2 Troncation et extension explicite

Une troncation ou extension explicite peut se faire à l'aide des appels système `truncate` et `ftruncate`[1] :

```
int truncate(char *name, off_t length);
int ftruncate(int fd, off_t length);
```

Comme beaucoup d'appels système agissant sur les fichiers, cet appel système existe en deux variantes : pour `truncate`, le fichier à tronquer ou étendre est spécifié à l'aide de son nom `name`, tandis que pour `ftruncate`, il est spécifié à l'aide d'un descripteur de fichier `fd`. Le paramètre `length` spécifie la nouvelle taille du fichier. S'il est inférieur à la taille actuelle, des données sont perdues (le fichier est tronqué) ; s'il y est supérieur, des zéros sont écrits à la fin du fichier (le fichier est étendu).

Ces appels système ne changent pas le pointeur de position courante, et retournent 0 en cas de succès, -1 en cas d'erreur (et alors `errno`… bon, vous avez compris).

1 Traditionnellement, ces appels servent à faire une troncation ; sur les Unix modernes, ils permettent aussi de faire une extension.

`O_TRUNC` Il est très courant de vouloir tronquer un fichier à 0 juste après l'avoir ouvert. Pour éviter de faire un appel à `ftruncate` après chaque ouverture, la troncation peut être combinée à cette dernière en positionnant le drapeau `O_TRUNC` dans le deuxième paramètre de `open` :

```
fd = open("alamakota", O_WRONLY | O_CREAT | O_TRUNC, 0666);
```

À la différence des drapeaux `O_EXCL` et `O_APPEND`, qui sont essentiels pour éviter des situations critiques dans certains cas, le drapeau `O_TRUNC` n'est qu'une abréviation.