Programmation système : programmation non bloquante

Juliusz Chroboczek

30 Décembre 2009

Les appels système `read`, `write`, lorsqu'ils sont appliqués à des tubes (ou des périphériques, ou des *sockets*) peuvent *bloquer* pendant un temps indéfini. Cette sémantique est souvent celle qui convient, mais elle pose problème lorsqu'on veut appliquer des *time-outs* aux opérations d'entrées-sorties, ou lire des données sur plusieurs descripteurs de fichiers simultanément.

Un descripteur de fichier peut être mis en mode *non-bloquant* en passant le *flag* `O_NONBLOCK` à l'appel système `open`. Une opération de lecture ou écriture sur un tel descripteur ne bloque jamais ; si elle devait bloquer, elle retourne -1 avec `errno` valant `EWOULDBLOCK` (« cette opération aurait bloqué »).

1 Changement des *flags* d'un descripteur

Les *flags* d'un descripteur obtenu à l'aide de l'appel système `open` sont affectés lors de l'ouverture. Souvent, cependant, les descripteurs sont obtenus à l'aide d'appels système qui ne permettent pas d'affecter les *flags*, notamment l'appel système `pipe` et, comme nous le verrons au deuxième semestre, l'appel système `socket`.

Les *flags* d'un descripteur existant peuvent être récupérés et affectés à l'aide de l'appel système `fcntl` :

```
int fcntl(int fd, F_GETFL);
int fcntl(int fd, F_SETFL, int value);
```

La première forme ci-dessus permet de récupérer les *flags* ; en cas de réussite, elle retourne les flags. La deuxième affecte les *flags* à la valeur passée en troisième paramètre.

Le *flag* `O_NONBLOCK`, qui nous intéresse, peut donc être positionné sur un descripteur de fichier `fd` à l'aide du fragment de code suivant :

```
rc = fcntl(fd, F_GETFL);
if(rc < 0)
    ...
rc = fcntl(fd, F_SETFL, rc | O_NONBLOCK);
if(rc < 0)
    ...
```

2 Attente active

Sur un descripteur non-bloquant, il est possible de simuler une lecture bloquante à l'aide d'une *attente active*, c'est à dire en vérifiant répétitivement si l'opération peut réussir sans bloquer :

```
while(1) {
    rc = read(fd, buf, 512);
    if(rc >= 0 || errno != EWOULDBLOCK)
        break;
}
```

Une attente active est plus flexible qu'un appel système bloquant. Par exemple, une attente avec *time-out* s'implémente simplement en interrompant l'attente active au bout d'un certain temps :

```
while(1) {
    rc = read(fd, buf, 512);
    if(rc >= 0 || errno != EWOULDBLOCK)
        break;
    if(time(NULL) >= debut + 10)
        break;
}
```

De même, une lecture sur deux descripteurs de fichiers `fd1` et `fd2` peut se faire en faisant une attente active simultanément sur `fd1` et `fd2` :

```
while(1) {
    rc1 = read(fd1, buf, 512);
    if(rc1 >= 0 || errno != EWOULDBLOCK)
        break;
    rc2 = read(fd2, buf, 512);
    if(rc2 >= 0 || errno != EWOULDBLOCK)
        break;
}
```

Comme son nom indique, l'attente active utilise du temps CPU pendant l'attente ; de ce fait, elle n'est pas utilisable en pratique. Les mitigations qui consistent à céder le processeur pendant l'attente (à l'aide de `sched_yield` ou `sleep`/`usleep`) ne résolvent pas le fond du problème.

3 L'appel système `select`

L'appel système `select` permet d'effectuer l'équivalent d'une attente active sans avoir besoin de boucler. L'appel `select` prend en paramètre des ensembles de descripteurs de fichiers (`fd_set`) et un *time-out* sous forme d'une structure `timeval`.

3.1 La structure `timeval`

Nous avons déjà vu le type `time_t`, qui représente un temps en secondes, soit relatif, soit absolu (mesuré depuis l'Époque), et l'appel système `time`, qui retourne un temps absolu représenté comme un `time_t`.

4BSD a introduit la structure `timeval`, qui représente un temps, absolu ou relatif, mesuré en secondes et micro-secondes :

```
struct timeval {
    time_t tv_sec;
    int tv_usec;
}
```

Le champ `tv_usec` représente les micro-secondes, et doit être compris entre 0 et 999 999.

Le temps absolu peut être obtenu à l'aide de l'appel système `gettimeofday` :

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

Le paramètre `tv` pointe sur un tampon qui contiendra le temps courant après l'appel. Le paramètre `tz` est obsolète, et doit valoir `NULL`.

3.2 Ensembles de descripteurs

Un ensemble de descripteurs est représenté par le type opaque `fd_set`. Les opérations sur ce type sont :

- `FD_ZERO(fd_set *set)`, qui a□ecte l'enseble vide à son paramètre ;
- `FD_SET(int fd, fd_set *set)`, qui ajoute l'élément `fd` à son deuxième paramètre ;
- `FD_ISSET(int fd, fd_set *set)`, qui retourne vrai si `fd` est membre de son deuxième paramètre.

3.3 L'appel système `select`

L'appel système `select` permet d'attendre qu'un descripteur de □chier parmi un ensemble soit prêt à e□ectuer une opération d'entrées-sorties sans bloquer, ou qu'un intervalle de temps se soit écoulé.

```
int select(int nfds,
           fd_set *readfds, fd_set *writefds,  fd_set *exceptfds,
           struct timeval *timeout);
```

Les paramètres `readfds`, `writefds` et `exceptfds` sont les ensembles de descripteurs sur lesquels on attend, et `timeout` est un temps relatif qui borne le temps pendant lequel on attend. Le paramètre `nfds` doit être une borne exclusive des descripteurs de □chiers contenus dans les trois ensembles.

Un appel à `select` bloque justqu'à ce qu'une des conditions suivantes soit vraie :

- un des descripteurs contenus dans `readfds` est prêt pour une lecture ; ou
- un des descripteurs contenus dans `writefds` est prêt pour une écriture ; ou

– un des descripteurs contenus dans `exceptfds` est dans une situation exceptionnelle (hors programme pour ce cours) ; ou
– un temps égal ou supérieur à `timeout` s'est écoulé.

L'appel à `select` retourne 0 dans ce dernier cas, et le nombre de descripteurs prêts dans les autres cas. Les descripteurs prêts sont alors positionnés dans les trois ensembles.

Dans tous les cas, la valeur de `timeout` est indé□nie (arbitraire) après l'appel.

3.4 Exemples

Une lecture bloquante sur un descripteur se simule en bloquant d'abord à l'aide de `select`, puis en e□ectuant une lecture :

```
fd_set readfds;
FD_ZERO(&readfds);
FD_SET(fd, &readfds);
rc = select(fd + 1, &readfds, NULL, NULL, 0);
if(FD_ISSET(fd, &readfds))
    rc = read(fd, buf, 512);
```

Comme dans le cas d'une attente active, il est facile d'ajouter un *time-out* :

```
struct timeval tv = {10, 0};
fd_set readfds;
FD_ZERO(&readfds);
FD_SET(fd, &readfds);
rc = select(fd + 1, &readfds, NULL, NULL, &tv);
if(FD_ISSET(fd, &readfds))
    rc = read(fd, buf, 512);
```

Il est aussi facile de généraliser cette attente à plusieurs descripteurs :

```
fd_set readfds;
FD_ZERO(&readfds);
FD_SET(fd1, &readfds);
FD_SET(fd2, &readfds);
rc = select((fd1 >= fd2 ? fd1 : fd2) + 1, &readfds, NULL, NULL, 0);
if(FD_ISSET(fd, &readfds))
    rc = read(fd, buf, 512);
```

3.5 Bug

La plupart des systèmes Unix ont le bug suivant : un appel `select` peut parfois retourner un descripteur de □chier qui n'est pas prêt. De ce fait, il est nécessaire de mettre tous les descripteurs de □chier en mode non bloquant même si on fait attention à ne jamais faire une opération bloquante que sur un descripteur qui a été retourné par `select`.