OS : Serveur multi-client

TP INFO-F201 : séance 9

Jérôme De Boeck, Guillerme Duvillié, Nassim Versbraegen

UlB

Implémentation d'une application client/serveur avec plusieurs clients

1 Multiplexage

Il peut être utile pour un serveur de recevoir des données sur un socket en même qu'accepter de nouvelles connexion. Autrement dit, il serait intéressant de pouvoir manipuler plusieurs sockets. Pour ce faire, on peut avoir recours à la fonction `select`, cette fonction permet d'écouter sur plusieurs sockets simultanément et d'être réveillé lorsque l'un d'entre eux devient actif.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

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

Le type `fd_set` représente un ensemble de descripteurs de fichiers. Ces ensembles peuvent être mis à jour avec les macros suivantes :

- `FD_SET(int fd, fd_set *set)` – ajouter `fd` à un ensemble
- `FD_CLR(int fd, fd_set *set)` – enlever `fd` d'un ensemble.
- `FD_ZERO(fd_set *set)` – vider un ensemble.
- `FD_ISSET(int fd, fd_set *set)` – tester si `fd` appartient à l'ensemble.

La fonction `select` écoute trois ensembles de descripteurs de fichiers : `readfds`, `writefds`, et `exceptfds` et attend que (au moins) l'un d'entre eux devienne actif. Après un appel réussi à la fonction, l'ensemble `readfds` contiendra les descripteurs de fichiers prêts à être lus et `writefds` ceux prêts à être écrits.

Le paramètre `n` doit être égal au plus grand des descripteurs de fichiers surveillés plus un. On a donc :

```
n = max (fd) + 1
```

Enfin le paramètre `timeval` peut être utilisé pour spécifier un temps maximum d'attente, s'il n'est pas mis à `NULL`, il s'agit d'un pointeur sur la structure suivante :

```
struct timeval {
    int tv_sec;     // seconds
    int tv_usec;    // microseconds
};
```

Notez que `select` permet de surveiller des descripteurs de fichier, ce qui veut dire qu'elle peut être utilisée pour surveiller notamment le descripteur de fichier relatif à l'entrée standard, en effet, les descripteurs de fichiers offrent une interface unifiée pour l'ensemble des entrées/sorties d'un système POSIX (Unix et partiellement Linux par exemple). Ainsi un descripteur de fichier est associé à l'entrée standard (`STDIN_FILENO`) et ce dernier peut être surveillé par la fonction `select`.

Exercice 1. *Modifier le programme perroquet client pour faire en sorte que la chaîne de caractère envoyée par le client est saisie par l'utilisateur au clavier.*

2 Les Mutexes

Les mutexes sont des structures de données permettant de mettre un place une sorte de sémaphores sut des ressources partagées entre plusieurs threads, visant à garantir un accès unique à ces dernières. Elles permettent donc d'éviter les problmèmes d'accès concurrentiels.

L'appel système permettant d'initialiser un mutex est le suivant :

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                       const pthread_mutexattr_t *restrict attr);
```

Le second paramètre peut être NULL pour initialiser un mutex avec les attributs du mutex par défaut. On peut également initialiser un mutex avec les valeurs par défaut en utilisant l'instruction :

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER
```

Une fois initialisé, il existe deux appels système pour verrouiller le mutex (demander l'accès à la zone critique d'exécution) :

```
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

Le premier est un appel bloquant, en d'autres termes, le thread se met en attente jusqu'à ce que la zone critique soit libérée, alors que le second retourne une erreur si le verrouillage du mutex n'est pas réalisable au moment de l'appel. Dans ce cas, la fonction retourne la valeur EBUSY. En cas de réussite, les deux appels retournent 0.

Pour déverrouiller e mutex (une fois que le traitement sur la ressource partagée est terminé), le thread doit appeler l'appel système :

```
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Cet appel est très important puisque la ressource critique ne sera accessible par les autres threads que lorsque le mutex aura été déverrouillé par le thread ayant verrouillé ce dernier.

Enfin l'appel système suivant permet de détruire un mutex :

```
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Exercice 2.

- *Ecrivez un code qui lancera trois threads. Le programme possédera une variable globale résultat initialisée à 4. Le premier thread devra itérativement multiplier résultat par 2, et ceci 26 fois d'affilée. Le second devra, en parallèle, ajouter 5 à la variable résultat, et ce, 170 fois itérativement. Le troisième devra finalement, en parallèle, soustraire (toujours itérativement) 3 à résultat 168 fois. De plus, vous devrez, à chaque itération, afficher le résultat intermédiaire. Exécutez le code plusieurs fois, que constatez-vous ?*
- *Ajoutez un mutex afin de verrouiller chaque série d'itérations afin d'afficher un résultat plus cohérent et stable durant celles-ci.*

3 Multi-client

Exercice 3. *À partir de votre serveur perroquet, écrire deux programmes serveur permettant de gérer plusieurs clients simultanément :*

1. *le premier en utilisant la programmation multi-thread,*
2. *le second en utilisant la programmation multi-processus.*