Travaux Dirigés Programmation Système: Feuille 7

Informatique 2ème année. ENSEIRB 2011/2012

—Denis Barthou - `dbarthou@enseirb.fr` —

Threads et réseau

L'objectif de ce TD de finir de voir les aspects threads et section critique, puis d'aborder la communication sur réseau par les sockets.

▸**Exercice 1.** *Lecteurs et écrivains (suite et fin)*

Écrire un programme qui lance deux threads, un lecteur et un écrivain, partageant un tableau A commun d'un million d'éléments.

L'écrivain parcourt le tableau et en incrémente tous les éléments, affiche un message puis fait `sleep(1)`*. Il effectue cette action 10 fois puis termine normalement.*

Le lecteur parcourt le tableau et calcule la somme de tous ses éléments et affiche un message. Si la somme n'est pas paire, il affiche un message, puis fait `sleep(1)`*. Il effectue cette action 10 fois puis termine normalement.*

1. *Tester le code avec un lecteur et un écrivain en mettant en place à l'aide de mutex (fonctions* `pthread_mutex_init(3)`*,* `pthread_mutex_lock(3)`*,* `pthread_mutex_unlock(3)` *et* `pthread_mutex_destroy(3)`*) un exclusion mutuelle sur la section critique (l'accès au tableau A).*
2. *On veut maintenant pouvoir lancer plus d'un thread lecteur. Plusieurs lecteurs peuvent accéder simultanément au tableau A. En revanche, lecteurs et écrivain sont en exclusion mutuelle. Modifiez votre programme pour que 4 lecteurs soient lancés dans 4 threads et qu'un écrivain soit lancé dans un thread.*
3. *On souhaiterait maintenant donner la priorité à l'écrivain : dès que l'écrivain attend l'entrée en section critique, les lecteurs ne peuvent pas lui passer devant et sont mis en attente, le temps que l'écrivain finisse sa mise à jour de A. Modifiez votre programme en ce sens, en utilisant (au moins) un autre sémaphore.*

▸**Exercice 2.** *Les bases d'un serveur*

Cet exercice a pour but d'ouvrir une partie de la connection réseau (celle du serveur, connection appelée socket).

Créez un programme qui établit attend une connection (qu'on établiera avec telnet) et quitte. Les étapes sont les suivantes :

1. *Création d'une socket avec* `socket(2)`*. Ouvrir une socket pour communiquer sur internet (IPv4, flag* `AF_INET`*) en mode connecté, duplex (flag* `SOCK_STREAM`*) et laissez le système choisir le protocole (flag 0). La socket sera fermée à la fin du programme par un appel à* `close(2)`*.*
2. *Initialiser une structure* `sockaddr_in` *qui contient les champs :*
   - `sin_family` *: utiliser* `AF_INET` *pour protocole internet*
   - `sin_addr.s_addr` *: utiliser* `INADDR_ANY` *pour que le système choisisse l'adresse internet de la machine*
   - `sin_port` *: le numéro du canal de communication, appelé port. C'est un entier quelconque,* > 1024*. Attention, utiliser les convertisseurs* `htons(3),...` *pour que le codage des entiers soit bien celui du réseau, pas celui de la machine.*
3. *Associer une adresse à la socket précédemment créée en passant la structure* `sockaddr_in` *et la socket à* `bind(2)`*.*
4. *Se mettre à l'écoute des connections avec* `listen(2)` *et spécifier le nombre d'entrées dans la file d'attente des demandes de connection.*
5. *Accepter (et attendre) une connection entrante avec* `accept(2)`*. Les deux derniers arguments seront mis à nul. Le descripteur de socket retourné peut être manipulé comme un descripteur de fichier.*
6. *Dès qu'une connection est acceptée (utiliser* `telnet` *pour établir une connection), fermer le descripteur de socket retourné par* `accept(2)` *et par* `socket(2)`*.*

▸**Exercice 3.** *Serveur echo*

Un serveur echo accepte les connections comme dans l'exercice précédent et renvoie au client tout ce qu'il a reçu, tant qu'il ne reçoit pas "quit". Quand il reçoit "quit", il ferme la communication avec le client et se remet en attente d'autres connections. Les échanges avec les clients qui se connectent se font grâce à `read(2)` *et* `write(2)` *sur le descripteur obtenu par* `accept(2)`*.*

- *Ecrivez à partir de l'exercice précédent un serveur echo.*
- *Lancez plusieurs clients avec telnet simultanément. Que se passe-t-il ?*
- *Modifiez le serveur à l'aide de* `fork(2)` *pour pallier ce problème.*

▸**Exercice 4.** *Serveur web*

Un serveur web accepte les connections comme dans l'exercice précédent. Le protocole est le suivant :

- *Le client (navigateur) lui envoie une chaine de caractères, contenant notamment la séquence "GET fichier" où fichier est le fichier à récupérer par le serveur web (un fichier html par exemple).*
- *Le serveur renvoie la chaine*
  "HTTP/1.0 200 OK\r\nContent-Type : text/html\r\n\r\n"
  suivie du contenu du fichier

Le type du fichier peut être adapté (ici `text/html`*) suivant qu'il s'agit d'un fichier html, jpg, png, ...*

- *Ecrivez un serveur web. Vous utiliserez* `stat(2)` *pour savoir si le fichier demandé existe et avoir sa taille, et* `mmap(2)` *pour le mettre en mémoire et ainsi l'envoyer avec* `write(2)`*.*
- *Réécrire le serveur pour qu'il soit multithreadé. Pour ce faire, on créera un nouveau thread pour traiter chaque connection.*
- *Modifier la solution précédente pour créer 8 threads dès le départ. Ces threads sont placés au début en attente, et ne sont activés que lorsqu'il y a des connections à traiter. C'est un exemple du problème producteur/consommateur : le producteur est le serveur, qui "produit" des requêtes à traiter, les threads sont les consommateurs et traitent ces requêtes. Adapter la solution à ce problème au programme.*