Systèmes d'exploitation : étude de cas
Seconde session – Proxy HTTP (le retour)

Geoffrey Miche
gmi@info.fundp.ac.be

Jean-François Wauthy
jfw@info.fundp.ac.be

2 juillet 2008

1 Proxy HTTP

Un serveur *proxy*, ou serveur mandataire, HTTP est un serveur qui fait suivre les requêtes HTTP de ses clients vers le serveur souhaité. L'utilisation d'un serveur *proxy* permet d'effectuer des opérations en plus d'un simple "routage". Un autre avantage d'un serveur *proxy* est le fait que l'on peut limiter l'accès à l'Internet à seulement son IP. Un serveur *proxy* peut aussi effectuer du filtrage, de l'authentification et cacher les données reçues afin de pouvoir servir plus rapidement une future requête similaire.

2 *Socket* non bloquant

Un *socket* est rendu non bloquant en précisant le *flag* `O_NONBLOCK` sur le *file descriptor* du *socket* à l'aide de la fonction `fcntl`. Le *socket* doit donc être configuré en non bloquant après sa création mais avant son utilisation.

Sur un *socket* non bloquant, les fonctions `connect`, `read` et `write` rendront directement la main même si la connexion n'est pas encore établie ou s'il n'y a rien à lire ou écrire. Afin d'éviter de devoir faire tourner une boucle active infinie sur tous les *sockets*, la fonction `select` permet de les "surveiller" et ne rend la main que si un des *sockets* reçoit des données ou est prêt à en envoyer.

3 Énoncé

On vous demande de réaliser un serveur *proxy* HTTP 1.1[1] utilisant **uniquement des appels réseaux non bloquants** ; si nécessaire, en rendant non bloquant des appels à des fonctions uniquement bloquantes (i.e. `gethostbyname`). Ce serveur pourra être déployé de manière chaînée avec d'autres serveurs *proxy*. Le serveur devra également être capable de charger des modules permettant de déterminer si une requête peut être acceptée par le serveur sur base des informations qu'elle contient. Enfin, et conformément à la logique d'un architecture non-bloquante, **un seul appel à `select`** sera utilisé.

Comme pour les TP précédents, il vous faudra aussi tenir compte de la sécurité dans votre code ainsi que de la libération systématique de toutes les ressources allouées dynamiquement durant son exécution (même si certaines ressources le sont à la fermeture du programme). Enfin, votre code devra **obligatoirement** pouvoir être compilé à l'aide de *gcc* avec les paramètres suivants : **`-ansi -pedantic -Wall -Werror`**.

[1] La RFC 2616 (`http://tools.ietf.org/html/rfc2616`) décrit en détails le protocole HTTP 1.1

3.1 Règles de filtrage

On vous demande également d'implémenter au minimum les 3 règles de filtrage suivantes :

1. Blocage de certains navigateurs ;
2. Blocage d'URL contenant au moins un mot-clé interdit ;
3. Blocage de pages sur base de leur nom de domaine. Par exemple, `www.info.fundp.ac.be` et `leibniz.info.fundp.ac.be` seront bloqués si `info.fundp.ac.be` est dans la liste des noms de domaine refusés. Par contre, les requêtes vers `www.fundp.ac.be` seront acceptées.

La configuration des différentes règles de filtrage sera stockée dans un fichier.

3.2 Bonus

En bonus, vous pouvez implémenter le support du cache comme demandé en première session. De plus, vous pouvez également implémenter au moins 3 règles de filtrage supplémentaires. Ces règles sont laissées à votre choix mais on vous demande de le motiver dans le rapport.

4 Date de remise

Un rapport détaillant clairement votre schéma de raisonnement, votre démarche de programmation, les résultats obtenus commentés, la description des difficultés rencontrées ainsi que la façon dont vous les avez surmontées est à remettre (avec les sources du TP en annexe) pour le **15 août 2008** au plus tard. Ce rapport ainsi que les sources du TP seront également remis par voie électronique.

Bon travail !