TP7 Client et proxy HTTP[1]

1 Le protocole HTTP

Le protocole HTTP est utilisé pour transférer des pages web entre un serveur (distant) et un client (votre programme). On utilisera la version 1.1 du protocole. La description présentée ici est très partielle mais devrait suffire pour le TP. Voir la partie documentations de référence pour tout complément d'information.

Un petit bibliothèque de routines utiles se trouvent à l'adresse suivante :

`http://www.di.ens.fr/~pouzet/cours/systeme/tp07/tp-07.tgz`

1.1 Requête HTTP

Pour demander la page `http://www.di.ens.fr/~pouzet/cours/systeme/index.html`, le client doit ouvrir une connexion TCP sur le port 80 vers `www.di.ens.fr` et envoyer les lignes de texte suivantes (`query.txt`):

```
GET /~pouzet/cours/systeme/index.html HTTP/1.1
Host: www.di.ens.fr
User-Agent: Mon Super Programme v0.01
Connection: close
ligne vide
```

Ces lignes ont la signification suivante :

- la première ligne indique le type de requête (`GET`), l'adresse absolue sur le serveur de la page web demandée (`/~pouzet/cours/systeme/index.html`) et la version du protocole utilisée (`HTTP/1.1`),
- la deuxième ligne rappelle le nom du serveur contacté (`www.di.ens.fr`),
- la troisième ligne permet à l'application cliente de s'identifier (facultatif mais poli; généralement, on y trouve quelque chose comme `Mozilla/5.0 (X11; U; Linux x86_64; fr; rv:1.8.1.3) Gecko/20070321 BonEcho/2.0.0.3`),
- la quatrième ligne indique que c'est au serveur de fermer la connexion quand il a fini d'envoyer la page (sinon, c'est au client de fermer la connexion),
- la dernière ligne est toujours vide ; elle indique la fin de l'en-tête.

En réalité, la ligne de requête peut être suivie de nombreuses autres lignes d'en-tête, toujours sous la forme `XXX: YYY`. Notez que le nom de l'en-tête `XXX` est insensible à la casse. Seule l'en-tête `Host:` est obligatoire. (Voir la partie 14 de la RFC 2616 pour la liste des en-têtes.)

Note importante : dans tous les protocoles textes d'Internet, les fins de ligne sont matérialisées par la séquence `\r\n`.

1. Sujet basé sur des TPs proposés en 2007 par Antoine Miné. Le texte a été créé par Louis Mandel.

1.2 Réponse positive

Une réponse du serveur à notre requête ressemblerait à ceci :

```
HTTP/1.1 200 OK
Content-type: text/html
ligne vide
<html><head>
...
</body></html>
```

Cette réponse consiste en trois parties :

- une ligne de statut commençant par le numéro de version du protocole (`HTTP/1.1`), suivi d'un code de retour numérique (`200` signifie que tout s'est bien passé) puis d'un texte (`OK`) explicitant la signification du code d'erreur,
- un nombre arbitraire de lignes d'en-têtes, terminées par une ligne vide (ici, on n'en a montré qu'une seule : `Content-type: text/html` qui indique le type MIME du document envoyé par le serveur : du texte au format HTML),
- le contenu de la page web demandée.

La fin de la page web est indiquée par la fermeture de la connexion de la part du serveur.

1.3 Réponse d'erreur

Si la page web n'existe pas, la réponse ressemblera à :

```
HTTP/1.1 404 Not Found
Content-type: text/html
ligne vide
<html><head><title>Not Found</title></head><body>
Sorry, the object you requested was not found.
</body><html>
```

Cette fois, le code est `404` qui signifie : page non trouvée. Notez que le serveur nous envoie tout de même une page web qu'un navigateur pourra choisir d'afficher. Ceci permet à l'administrateur du site de configurer la manière dont le message d'erreur est présenté à l'utilisateur. (La liste complète des codes de statut se trouve dans la partie 10 de la RFC 2616.)

1.4 Autres réponses intéressantes

Dans certains cas, le serveur indique que la page a été déplacée temporairement ou définitivement (codes `301`, `302`, `303`, `305` et `307`) à une autre adresse. Celle-ci est indiquée par une en-tête de la forme `Location:` nouvelle adresse.

2 Récupérer une page web

Le but de cette partie est de faire un programme qui télécharge une page web (dont l'adresse est donnée en argument) et affiche son contenu (hors en-têtes) à l'écran. L'adresse de la page web est donnée sous la forme d'une URL `protocole://serveur:port/chemin` où `protocole` vaut toujours `http`, `serveur` est une adresse texte ou numérique, la partie `:port` est facultative (la valeur par défaut `80` sera utilisée si le port est omis) et chemin indique le chemin de la page sur le serveur (le séparateur de répertoire est `/`).

Question 1. Écrire une fonction `make_addr: string -> int -> Unix.sockaddr` qui prend en paramètre un nom de machine et un numéro de port et retourne une adresse.
(Unix.gethostbyname, Unix.ADDR_INET)

Question 2. Écrire une fonction `run_client: (file_descr -> 'a) -> sockaddr -> 'a` telle que `run_client f addr` crée une socket qui se connecte à l'adresse `addr` et qui appelle la fonction `f` sur le descripteur de fichier correspondant à la socket une fois connectée.
(Unix.socket, Unix.PF_INET, Unix.SOCK_STREAM, Unix.connect)

Question 3. Écrire le programme qui fait une requête HTTP sur une URL donnée en argument.

Prenez soin à rapporter à l'utilisateur les erreurs rencontrées (serveur inexistant, connexion refusée, page non trouvée, etc.). (Unix.write/Unix.send, Unix.read/Unix.recv, Unix.in_channel_of_descr, Unix.out_channel_of_descr, flush)

3 Fonctionnement d'un proxy HTTP

Un proxy (aussi appelé serveur mandataire) est un serveur capable de relayer des pages web entre un client et un serveur HTTP.

Au lieu de se connecter directement au serveur indiqué par l'utilisateur, le client se connecte au proxy. Celui-ci contacte le serveur réel, télécharge la page demandée et la retransmet au client. Vu du côté du client, le proxy agit comme un serveur HTTP. Vu du côté du serveur, le proxy agit comme un client HTTP.

Quand le client et le serveur sont séparés par un pare-feu qui interdit toute connexion directe, l'emploi d'un proxy (qui tourne sur le pare-feu) est indispensable.

Généralement, un proxy ne se contente pas de relayer de façon transparent toutes les requêtes et les réponses HTTP. Parmi les utilisations possibles d'un proxy HTTP on compte :

- mettre en cache les pages les plus fréquemment demandées,
- faire des statistiques sur les pages accédées,
- filtrer l'accès à certaines pages web,
- imposer à l'utilisateur de s'identifier,
- protéger l'utilisateur en rendant anonymes les connections (e.g.: en maquillant les en-têtes HTTP, en supprimant les cookies, en routant les requêtes à travers plusieurs proxy successifs),
- établir un tunnel qui compresse les transactions HTTP, les chiffre (e.g.: SSL) ou les convertit dans un autre protocole,
- modifier les pages web à la volée.

3.1 Configuration des clients HTTP

La plupart des clients HTTP peuvent être configurés pour utiliser un proxy. Par exemple:

- pour Firefox, allez dans `Édition->Préférences`, icône `Avancé`, onglet `Réseau`, bouton `Paramètres`, champ `Configuration manuelle du proxy` ;
- pour `lynx` et `wget`, il faut de mettre la variable d'environnement `http_proxy` à `http://localhost:8080` ;
- pour `links`, on utilisera l'option `-http-proxy localhost:8080` en ligne de commande.

3.2 Protocole de proxy

Le protocole HTTP supporte explicitement l'utilisation de proxys grâce à des en-têtes dédiés. Une requête d'un client à un proxy ressemble à ceci :

```
GET http://www.di.ens.fr/~pouzet/cours/systeme/index.html HTTP/1.1
Host: www.di.ens.fr
Proxy-connection: keep-alive
autres en-têtes
...
ligne vide
```

On note les différences suivantes avec une requête classique HTTP 1.1 :

— `GET` spécifie toujours l'URL complète de la page, y compris le nom de protocole (`http://`) et le nom de serveur (`www.di.ens.fr`),

— l'en-tête `Host` indique le nom (et éventuellement le port) du serveur de la page recherchée, différent du nom du serveur proxy contacté,

— l'en-tête facultatif `Proxy-connection` est utilisé à la place de `Connection` pour indiquer si le serveur doit maintenir la connexion ouverte après avoir servi la page (`keep-alive`, par défaut si l'en-tête est absente) ou la fermer (`close`).

On rappelle que les en-têtes sont insensibles à la casse, que les lignes sont toutes terminées par `\r\n` et que la fin de l'en-tête est indiquée par une ligne vide.

4 Questions préliminaires

Question 4. Écrire une fonction `run_server` du type suivant :

```
(Unix.file_descr -> Unix.sockaddr -> 'a) -> Unix.sockaddr -> unit
```

L'appel `run_server service addr` doit créer une socket d'écoute sur l'adresse `addr` (Unix.socket, Unix.bind, Unix.listen). Ensuite, dans une boucle infinie, elle attend une nouvelle connexion (Unix.accept), traite la requête avec la fonction `service` qui est appelée avec comme paramètre le descripteur de fichier correspondant à la socket créée pour communiquer avec le client et l'adresse du client.

Question 5. Écrire une fonction `cat_service: file_descr -> sockaddr -> unit` qui affiche sur la sortie standard tout ce qui est écrit sur le descripteur de fichier donné en paramètre.

Question 6. Utiliser vos fonctions `run_server` et `cat_service` pour afficher les requêtes faites par différents clients web (Firefox, wget, etc.) lorsqu'ils communiquent par un proxy.

5 Proxy transparent

Question 7. Dans un premier temps, vous programmerez uniquement la partie serveur du proxy HTTP. Votre proxy se contentera de répondre une erreur `403 Forbidden` à toutes les requêtes du client, sans chercher à contacter de serveur web.

Question 8. Ajoutez maintenant la partie client afin de réaliser un proxy HTTP transparent. Celui-ci retransmet chaque requête au serveur indiqué dans l'en-tête `Host` et renvoie la réponse au client. Testez votre proxy avec plusieurs clients HTTP (Firefox, Konqueror, Lynx, Links, wget, etc). Pensez à bien gérer le cas où le serveur demandé par le client n'est pas joignable (`502 Bad Gateway`). Pensez également à forcer le mode `Connection: close` afin que le serveur ferme la socket à la fin de la réponse.

6 Documentations de référence

Voici des pointeurs sur les documentations officielles qui décrivent tous les aspects de HTTP et du HTML.

- la RFC 2396 (`http://www.ietf.org/rfc/rfc2396.txt`) définit la syntaxe des URLs (et, plus généralement, des URIs); voir également la RFC 2732 (`http://www.ietf.org/rfc/rfc2732.txt`) pour les URLs avec adresses IPv6 ainsi que la RFC 1808 (`http://www.ietf.org/rfc/rfc1808.txt`) qui donne un algorithme pour rendre canonique les URLs,
- la RFC 2616 (`http://www.ietf.org/rfc/rfc2616.txt`) définit le protocole HTTP 1.1,
- le format HTML 4.01 (`http://www.w3.org/TR/html401/`) est établi par le World Wide Web Consortium (W3C).