Projet – Base de données d'images

1 Description

On souhaite mettre au point une base de données d'images. Cette base est constituée de 3 entités distinctes :

- le serveur d'images : il conserve les images, stockées sous forme de fichiers JPEG ou autres formats. Pour la simplicité, on considérera (sans le vérifier) qu'une image a un nom unique dans la base.
- le serveur de tags : à chaque image sont associés zéro, un ou plusieurs « tags », un tag étant une simple chaîne de caractères (par exmple : « vacances à la plage », ou « lynx dans les vosges »).
  Pour la simplicité, on limitera les tags à des chaînes de caractères quelconques jusqu'à 64 octets. Un même tag peut être associé à plusieurs images.
- les clients : ceux-ci connaissent les différents serveurs (adresse IPv4 ou IPv6 ou nom, numéro de ports) et peuvent :
  - ajouter, remplacer ou supprimer une image
  - ajouter, remplacer ou supprimer un tag sur une image
  - rechercher les noms des images correspondant à un ou plusieurs tags
  - rechercher les tags associés à une image
  - récupérer une image étant donné son nom

2 Programmes

On demande de rédiger les programmes suivants :

- `image` *`port répertoire`*
  Ce programme est le serveur d'images : il écoute les requêtes sur le port TCP indiqué et gère les images stockées dans le répertoire indiqué.
- `tag` *`port répertoire`*
  Ce programme est le serveur de tags : il écoute les requêtes sur le port UDP indiqué et gère les tags stockés dans le répertoire indiqué.
- `client`
  Ce programme est le client. Il comprend plusieurs syntaxes :
  - `client image add` *`image`* : ajoute (ou remplace) l'image indiquée par *`image`* (qui est à la fois le nom de l'image et le nom du fichier fourni au serveur d'images)
  - `client image del` *`image`* : supprime l'image indiquée par *`image`* sur le serveur d'images. Les tags correspondants doivent également être supprimés sur le serveur de tags.
  - `client image tag` *`image`* : récupère sur la sortie standard les tags associés à *`image`* (qui doit exister)
  - `client image get` *`image`* : récupère sur la sortie standard l'image indiquée par *`image`* depuis le serveur d'images
  - `client tag add` *`image tag`* : ajoute (ou remplace) le tag indiqué sur l'image (qui doit exister)
  - `client tag del` *`image tag`* : supprime le tag indiqué sur l'image (qui doit exister)
  - `client tag image` *`tag ... tag`* : récupère sur la sortie standard les noms des images associées aux tags indiqués

  Pour identifier les serveurs, le programme `client` utilise un fichier contenant les coordonnées de chaque serveur (adresse IPv4 ou adresse IPv6 ou nom DNS, ainsi que le numéro de port).

3 Exemple de session

On trouvera ci-après un exemple (fictif) d'utilisation du programme `client` :

```
$ ls
lynx1.jpg lynx2.jpg
```

```
$ ./client image add lynx1.jpg                  # ajout d'image
Image 'lynx1.jpg' stockee
$ ./client image add lynx2.jpg                  # ajout d'image
Image 'lynx2.jpg' stockee
$ ./client image add lynx2.jpg                  # remplacement d'image
Image 'lynx2.jpg' stockee

$ ./client image tag mamie.jpg                  # liste les tags associés à mamie.jpg
plage
ete 2018
$ ./client tag image plage "ete␣2018"           # liste les images ayant les 2 tags
mamie.jpg
papy.jpg
tonton.jpg
$ ./client image del tonton.jpg                 # supprime également les tags associés à tonton.jpg
$ ./client image get papy.jpg > xxx.jpg         # récupère l'image et la place dans xxx.jpg

$ ./client tag add lynx1.jpg "lynx␣vosges"      # ajoute un tag à une image
Tag 'lynx␣vosges' ajoute a lynx1.jpg
$ ./client tag add lynx1.jpg "fevrier␣2021"
Tag 'fevrier␣2021' ajoute a lynx1.jpg
$ ./client tag del lynx1.jpg "fevrier␣2021"     # ooops, j'ai mis un mauvais tag
Tag 'fevrier␣2021' supprime sur lynx1.jpg
$ ./client tag add lynx1.jpg "fevrier␣2020"     # on remet le bon tag
Tag 'fevrier␣2020' ajoute a lynx1.jpg
```

4 Spécification des protocoles

Cette section décrit les protocoles que vous devez implémenter dans le cadre de ce projet.

4.1 Protocole image

Le serveur d'images écoute plusieurs types de requêtes, identifiées par la valeur du premier octet :

Requête	Signification	Paramètres	Réponse
0	lister les images présentes	(rien)	liste de noms
1	tester l'existence d'une image	nom	erreur
2	envoyer une image vers le serveur	nom et contenu	erreur
3	récupérer une image depuis le serveur	nom	contenu
4	supprimer une image sur le serveur	nom	erreur

Les différents objets manipulés (dans la requête ou la réponse) ont les formats suivants :

Objet	Format
nom	1 octet pour la longueur du nom ($< 2^8$ octets) suivi par le nom
erreur	1 octet pour la longueur du message ($< 2^8$ octets) suivi par le message
contenu	4 octets pour la taille ($< 2^{32}$ octets) suivis par le contenu
liste de noms	2 octets pour le nombre de noms ($< 2^{16}$ noms) suivis par les noms

Pour les requêtes susceptibles de renvoyer une erreur, la réponse est soit un message vide (absence d'erreur) ou non vide (message d'erreur). Pour la récupération d'image, si l'image n'existe pas, le contenu renvoyé doit être vide.

Les requêtes sont envoyées avec le protocole TCP. La réponse à chaque requête est envoyée dans la même connexion que la requête. Plusieurs requêtes (et donc les réponses associées) peuvent utiliser la même connexion TCP.

4.2 Protocole tag

Le serveur de tags écoute plusieurs types de requêtes identifiées par le premier octet :

Requête	Signification	Paramètres	Réponse
0	lister les tags associés à une image	image	liste de tags
1	lister les images associées à un ensemble de tags	liste de tags	liste d'images
2	associer un tag à une image	image et tag	(rien)
3	supprimer un tag d'une image	image et tag	(rien)

Les différents objets manipulés (dans la requête ou la réponse) ont les formats suivants :

Objet	Format
tag	1 octet pour la longueur ($< 2^8$ octets) suivi par le tag
image	1 octet pour le nom ($< 2^8$ octets) suivi par le contenu
liste d'images ou de tags	2 octets pour le nombre ($< 2^{16}$) suivis par les images ou les tags

Le protocole tag repose sur UDP. Pour simplifier, on supposera que chaque requête ou réponse tient dans un seul message (on ne vérifiera pas). On limitera toutefois l'attente d'une réponse par un client pour traiter le cas d'une panne sur le serveur.

5 Travail demandé

Vous devez implémenter les programmes mentionnés, en respectant scrupuleusement les protocoles définis ci-dessus. Vos programmes doivent être rédigés en langage C à l'aide des *sockets*, sans utiliser d'autre mécanisme ou bibliothèque de programmation réseau. Ils doivent être compilables sur la machine de référence `turing.unistra.fr` avec les options `-Wall`, `-Wextra` et `-Werror`, et exécutables sur cette même machine.

Vous rédigerez un rapport décrivant votre implémentation, les structures de données mises en œuvre, et les limitations et les extensions éventuelles que vous aurez réalisées.