EPFL CS212 : ImgStore – Système de fichiers orienté images — 12 : webserver 2 : `read`, `insert` et `delete`

E. Bugnion & J.-C. Chappelier EPFL

Rev. 2021.05.25 / 1

Table des matières

Introduction

L'objectif de cette semaine est de compléter le serveur web en implémentant les fonctions manquantes :

— le rajout d'une image (« *upload*») ;
— la lecture d'une image, dans la résolution demandée (« *download* ») ;
— la suppression d'une image.

Matériel fourni

Aucun matériel supplémentaire n'est fourni cette semaine, vous vous basez sur votre travail des semaines précédentes.

Travail à faire

Récupération des arguments

Le serveur web doit implémenter différents type de commandes, et chaque commande a ses propres arguments. Le format générique d'un URI (« *Uniform Resource Indicator* ») est :

`http://servername:port/dir1/dir2/dir3/file?arg1=foo&arg2=bar`

Par exemple, l'URI pour charger une image sera :

`http://localhost:8000/imgStore/read?res=orig&img_id=chaton`

Nous devons donc parser les arguments de telles URIs pour nos commandes. La bibliothèque `mongoose` sépare déjà le « *servername* » et « *port* » de l'URI. Dans votre « *handler mongoose* » (fonction chargé de gérer une commande, p.ex. `handle_read_call()`), la structure de type `struct mg_http_message` contient les champs `uri` et `query` qui correspondent respectivement à `/imgStore/read` et à `res=orig&img_id=chaton` dans l'exemple ci-dessus. Pour récupérer la valeur d'un argument, utilisez `mg_http_get_var()` sur la `query`. Ce qui n'est pas dit dans la documentation, c'est que cette fonction ne lit pas plus que `len` (dernier paramètre) **moins 1** `char`.

Erreurs HTTP

Nous vous recommandons également d'écrire une routine qui va retourner un code d'erreur HTTP — à utiliser si les arguments de la `query` ne correspondent pas à l'API spécifiée, ou si la requête ne peut être traitée avec succès.

Cette méthode appellera `mg_http_reply()` pour retourner :

- le code d'erreur HTTP 500 ;
- suivi du message d'erreur (tableau `ERROR_MESSAGES`), préfixé de « Error: », p.ex. `"Error: Invalid image ID"`.

Le prototype de cette méthode doit être :

```
void mg_error_msg(struct mg_connection* nc, int error);
```

Lecture

Implémentez une fonction `handle_read_call()`, équivalent de `handle_list_call()` mais pour le préfixe d'URI `/imgStore/read`.

Comme expliqué plus haut, cette fonction doit appeler `mg_http_get_var()` pour récupérer les arguments suivants :

- `res` : la version texte de la résolution d'image demandée ; à convertir ensuite avec `resolution_atoi()` (cf la commande `read` de votre interpréteur de commande des semaines précédentes) ;
- `img_id` : l'identifiant de l'image dans la base (son « nom »).

Ces deux paramètres sont obligatoires, mais l'ordre n'est pas important.
Exemple d'URI :

```
http://localhost:8000/imgStore/read?res=orig&img_id=chaton
```

Appelez ensuite la fonction `do_read()` avec les bons arguments.

En cas de succès, envoyez (avec `mg_printf()`) la réponse HTTP au format suivant :

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
Content-Length: <XXX>
```

La longueur de `Content-Length` doit être la taille de l'image (en octets). (Note : les lignes ci-dessus sont, comme toujours, terminées par `"\r\n"`.)
Utilisez ensuite la fonction `mg_send()` pour envoyer le contenu de l'image elle-même.

En cas d'erreur, appelez la fonction `mg_error_msg()` que vous avez implémenté ci-dessus.

Et pensez à bien désallouer la mémoire dynamique qui aurait été allouée par le programme...

Effacer une image

Implémentez la fonction `handle_delete_call()` devant répondre aux requêtes HTTP d'URI `/imgStore/delete`. Cet URI n'attend qu'un seul argument : `img_id`.

Une fois l'argument (valide) récupéré, appelez la fonction `do_delete()`. En cas de succès, retourner la réponse HTTP qui rechargera la page `index.html` :

```
HTTP/1.1 302 Found
Location: http://<URL>/index.html
```

où « `<URL>` » est l'adresse HTTP utilisée par le serveur.
En cas d'erreur, utilisez l'usuel `mg_error_msg()`.

Insertion

Implémentez enfin la fonction `handle_insert_call()`, la plus complexe, pour traiter les commandes d'URI `/imgStore/insert`.

La logique d'insertion est différente de celle qui retourne la liste (`list`) ou une image (`read`). En effet, l'insertion utilise la commande `HTTP POST`, alors que les deux autres utilisent `HTTP GET`. En gros, un `GET` contient tous les arguments dans l'URI, alors qu'un `POST` a des arguments additionnels en plus de l'URI. En particulier, la commande `/imgStore/insert` utilise un `POST` pour le contenu même de l'image à insérer.

Afin d'éviter des surcharges de la RAM du server, les gros fichiers sont en général envoyés morceau par morceau (« *chunk* ») en plusieurs `POST` successifs ; et c'est ce que nous avons fait dans `index.html`. Cette fonction `handle_insert_call()` aura donc essentiellement deux « mode » de fonctionnement :

— collecter les morceaux tant qu'il y en a ;
— faire l'insertion dans l'image store lors de la réception du dernier « morceau », lequel est en fait un morceau vide (`len` du `body` est à 0).

Pour stocker les morceaux, il suffit simplement d'appeler la fonction `mg_http_upload()`. Même si ce n'est pas idéal pour des raisons de sécurité/confidentialité, on pourra dans ce projet utiliser `/tmp` comme lieu de stockage temporaire.

Lors de la réception du dernier « morceau » (vide, `len` du `body` à 0), récupérer les paramètres `name` et `offset`. `name` contient le nom de l'image, que l'on utilisera comme identifiant pour l'insérer dans la base, et `offset` contient la taille de l'image (sous forme d'une chaîne de caractères ; il faut donc la convertir).

Allez ensuite lire le fichier de stockage temporaire (il a aussi le nom contenu dans `name`), puis appelez ensuite `do_insert()`.

En cas d'erreur, faites attention de bien retourner un message d'erreur adapté (en utilisant `mg_error_msg()`). En cas de succès, procédez comme avec `delete` pour réafficher la page d'index.

Finalisation

N'oubliez pas d'ajouter vos trois nouveaux URI au `imgst_event_handler()`.

De plus, puisque le traitement des images utilise (indirectement) la bibliothèque VIPS, il ne faudra pas oublier de la démarrer (`VIPS_INIT`) au lancement du serveur, et de la fermer (`vips_shutdown()`) à son arrêt.

Tests

Pour tester votre serveur web, comme la semaine passée : lancez simplement votre `imgStore_server` depuis un répertoire contenant le `index.html` fourni, puis ouvrez `http://localhost:8000/` dans un navigateur Web. Vous devriez obtenir quelque chose comme ça (cela dépend de la imgStore avec laquelle vous lancez votre serveur) :

— Cliquez sur une croix rouge à droite pour faire un `delete`.
— Cliquez sur une image pour la voir en taille d'origine (`read`).
— Cliquez sur le texte « *Click here to upload* » pour ajouter un fichier (`insert`).

Vous pouvez aussi tester directement les URI, comme p.ex. : `http://localhost:8000/imgStore/read?res=smal` pour tester la résolution « small ».

Enfin, il y a toujours le `make feedback` qui est à disposition (tests effectués via `curl`).

Rendu

Comme la semaine passée, vous n'avez pas à rendre tout de suite le travail de cette semaine. Celui-ci ne sera à rendre qu'à la fin (délai : le dimanche 06 juin 23:59) en même tant que tout le reste.