I) Gestion d'un index téléphonique par RPC

ESIEE - Laurent PERROTON

L'objectif du TP est de développer un service RPC (serveur et client) pour un index téléphonique disponible sur un serveur : recherche du numéro à partir du nom, ajout d'un nouveau numéro. L'index a la structure d'un fichier ASCII `index.data` contenant des lignes `"nom telephone"` de la structure suivante dans `index.h` :

```
struct ligne { char *nom; int tel; }
```

Les squelettes sources et Makefile sont accessibles sur `http://www.esiee.fr/~perrotol`

1 Client/Serveur haut niveau

Le code des fonctions de recherche et d'ajout dans l'index vous est fournis dans le source `index_local.c`.

`ligne * recherche_1 (char **nom)` prends comme argument un pointeur sur une chaîne de caractère contenant le nom de la personne dont on souhaite retrouver l'entrée dans le fichier, et retourne l'adresse d'une **structure** ligne contenant le nom et le numéro de téléphone trouvé. Si le nom n'est pas dans le fichier, la chaîne est **vide** et le numéro de téléphone égale à **0**.

`int * ajout_1 (ligne * a_ajouter)` ajoute une ligne **si elle n'est pas déjà présente** dans le fichier index. Elle renvoie 0 en cas d'échec, 1 sinon.

1) Développez les filtres XDR des types nécessaires dans un **fichier source séparé** `index_xdr.c` Attention à la gestion de l'allocation mémoire de façon à ne pas saturer le système.

2) Développez le client RPC et le serveur RPC haut niveau dans `index_clnt.c` et `index_svc.c` **en apportant le minimum de modification aux sources originaux** `index_local.c` et `index.c`.

Cette convention de nommage des sources vous permet d'utiliser le `Makefile` et de compiler les executables client et serveurs en tapant simplement `make`.

2 Fonction de listage

Rajoutez un fonction de listage de l'index en utilisant une **structure de liste chaînée**.

3 Fonction d'arrêt du serveur : appels RPC *one-way (bas niveau)*

Vous allez adjoindre au serveur une fonction d'arrêt *télécommandé* `void *stop_1 (char ** mot_de_passe)` à travers une procédure RPC. Testez avec l'API haut niveau, puis avec l'API bas niveau. Le serveur renvoie-t-il une réponse au client ? quelles sont les conséquences ? (tester en UDP et en TCP) Quels sont les services RPC référencés auprès du portmap ? Si les précédents points amènent des problèmes, proposer des solutions.

4 Requête avec Call-Back *(optionnel)*

On souhaite faire des requêtes non bloquantes au niveau du client. On utilisera la méthode du **Call-Back**. Il vous faudra écrire une première fonction qui **émettra la requête**. Ceci peut se faire sous forme d'un appel RPC conventionnel, c'est à dire bloquant. Le client recevra alors un *acknowledge* de sa requête. La véritable réponse à sa requête lui sera communiqué par le serveur sous forme de l'exécution d'une procédure RPC temporaire que le client aura crée pour recevoir cette réponse. Remarquez bien que pour ce faire, **la requête du client doit contenir une identification de ce service temporaire pour que le serveur puisse l'appeler** ; en conséquence, le format des requêtes est différent des précédentes.

II) Gestion d'un index téléphonique par RPC (rpcgen)

ESIEE - Laurent PERROTON

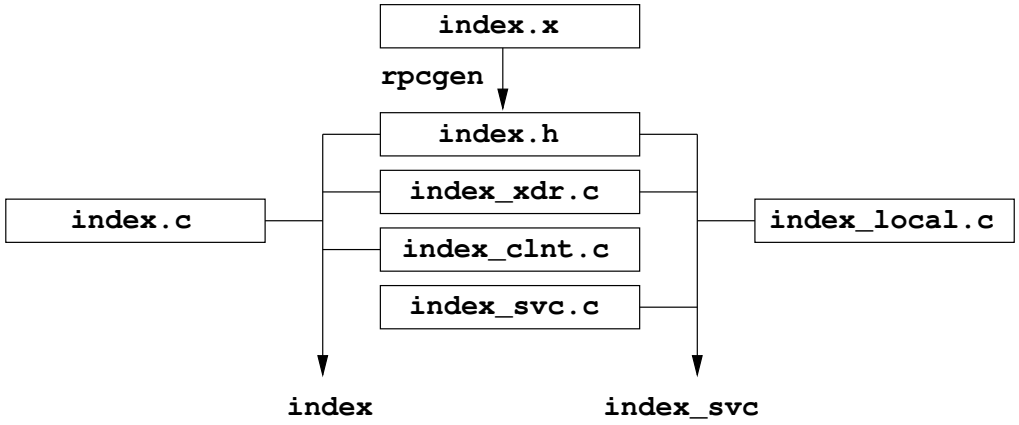
L'objectif du TP est de poursuivre le développement du service RPC de gestion d'annuaire téléphonique avec `rpcgen`.

1 Remarques

- Il est conseillé de faire une sauvegarde de vos sources pour les conserver avant de commencer à utiliser `rpcgen` car celui-ci va automatiquement les re-générer et donc écraser les précédents.
- Repartir des sources `index.c` et `index_local.c` à télécharger sur `http://www.esiee.fr/~perrotol`
- Dans le `Makefile`, enlevez les `#` devant les lignes qui concernent `rpcgen` pour activer son invocation et la génération automatique des sources.

2 `rpcgen`

Ecrire un fichier de spécifications RPCL `index.x` pour implémenter les services RPC de base de gestion d'annuaire téléphonique : recherche, ajout, faire afficher un message au serveur. L'objectif est de réaliser une application client/serveur en minimisant les modifications à apporter aux sources `index.c` et `index_local.c`



3 Services complémentaires

Rajouter les services suivant en utilisant `rpcgen` :

liste de l'index : fonction qui renvoie une liste chaînée des entrées de l'index. Comment doit être faite l'allocation mémoire de la liste **sur le serveur et sur le client** pour éviter toute fuite de mémoire entre plusieurs appels successifs ?

arrêt du serveur : fonction d'arrêt du serveur *(appel one-way non bloquant)*. Comment réalise-t-on un appel non bloquant avec `rpcgen` ? Quel type de retour faut il mettre dans le fichier de spécification `index.x` ? Observez le prototype et le code généré par `rpcgen`.

Quelles sont les avantages et les inconvénients d'utiliser rpcgen pour rajouter ces services ?

4 Recherche par Call-back *(prog. bas niveau)*

On souhaite faire des requêtes **non bloquantes** au niveau du client avec la méthode du *call-back*.

Le client crée un serveur *RPC temporaire* et émet une requête vers le serveur principal. Le serveur principal reçoit la requête du client et renvoie un accusé réception au client qui peut alors continuer ses activités. Le serveur principal traite la requête et appel le serveur temporaire du client (sous forme d'un appel RPC) pour renvoyer la réponse au client. Attention : le format des requêtes clientes doit inclure toutes les coordonnées (machine, numéro de service, version, procédure) pour que le serveur puisse le rappeler.

Implémenter cette fonctionnalité en utilisant la programmation bas niveau. Identifier les risques d'interblocages entre client et serveur. Est-il possible d'implémenter le call-back complètement avec `rpcgen` ou faut il programmer le client/serveur en bas niveau ?