TP ONC RPC

Le TP est noté ! vous devrez remettre la partie 4 sur la zone Moodle associée d'ici le 23 octobre minuit. Vous travaillerez en binôme en précisant sur le compte rendu vos 2 noms.

Objectifs pédagogiques associés :

Le but du TP est d'illustrer le modèle de programmation client-serveur offert par ONC RPC et de voir comment une application répartie peut être dérivée à partir de la version séquentielle.

Les parties 1, 2 et 3 sont à réaliser en première partie du TP (soit 1h15)
La partie 5 est optionnelle et vous trouverez sur Moodle un solution … la difficulté principale est liée à l'utilisation « pointue » de pointeurs en C….

Important – à lire avant de commencer:

- Vous pouvez faire ce TP sur Linux (de préférence) et votre PC (aussi de préférence)
- Sur le Mac la génération du Makefile (commande *rpcgen -Sm*) n'est pas disponible. Vous devrez donc obtenir le fichier Makefile autrement ou vous en passer.
- Si vous êtes sur votre machine Linux, il est possible que vous ne puissiez pas enregistrer un serveur sans les privilèges de « root ». ceci est lié à une mesure de sécurité de certaines distributions Linux. Si tel est le cas, exécutez la commande « rpcbind -i »

Partie 1: client serveur et calculatrice

Le fichier calc.x fournit une spécification de service de type RPC en vu de la définition d'une calculatrice. Cette calculatrice est « à pile » et comme vous pouvez le constater ne prend qu'un seul paramètre.

Q1 : Quels sont les services qui sont définis dans cette interface « calc.x » ?

Lancez ensuite la commande Unix :

```
rpcgen calc.x
```

À la suite de cette opération, des fichiers `calc.h`, `calc_clnt.c` etc sont générés et ne sont pas à modifier à ce stade.

Q2 : quelles sont les fonctions et déclarations contenues dans ces fichiers ? expliquez à partir d'un des services (ADD par exemple), leurs rôles respectifs.

Générer ensuite le Makefile et les fichiers applicatifs :

```
rpcgen –Sm calc.x > Makefile
rpcgen -Sc calc.x > calc_clnt_main.c
rpcgen -Ss calc .x > calc_svc_proc.c
```

```
Q3 : quelle est la nature du code généré pour les 2 derniers
fichiers ? regardez en particulier les sections « INSERT YOUR CODE
HERE ». A quoi correspondent-elles ?
```

```
On voit bien ici que le compilateur RPCL est capable de générer tout
une série de fichiers et ceci à partir d'un seul fichier calc.x.

Modifiez maintenant le numéro de version dans le fichier calc.x
Quels sont les impacts de ces modifications sur les fichies générés
(pour cela il faudra recompiler les fichiers).
```

Partie 2 : vers une calculatrice à pile

Les fichiers *calc_clnt_main.bon* et *calc_svc_proc.bon* sont des exemples de code écrits C d'un client et d'un serveur. Il nous faudra ensuite modifier leur suffixe.

Récupérer les fichiers que vous avez générés dans la partie 1 *calc_clnt_main.bon* et *calc_svc_proc.bon* et les renommer respectivement en *calc_clnt_main.c* et *calc_svc_proc.c*. à l'aide des commandes suivantes :

```
mv calc_clnt_main.bon calc_clnt_main.c
mv calc_svc_proc.bon calc_svc_proc.c
```

Examiner le code et comment le serveur estmis en œuvre. Quels sont les protocoles de communication qu'il peut gérer ?

Compléter ensuite le fichier Makefile afin qu'il référence les fichiers utilisés dans ce projet en modifiant les deux lignes suivantes à l'aide des caractères en italiques :

```
SOURCES_CLNT.c = calc_clnt_main.c
...
SOURCES_SVC.c = calc_svc_proc.c
```

Compilez le tout avec la commande ***make*** et lancez dans deux terminaux les exécutables obtenus (de type client et l autre serveur) ; un exécutable sera lancé sur p chaque terminal Linux. Testez votre application en fonction du code que vous avez analysé.

Q4 : quelles sont les séquences de traitement et d'appels qu'un serveur réalise lorsqu'un message RPC CALL arrive ? Donnez des explications et illustrez le.

Q5 : regarder la documentation en ligne, et précisez les arguments de la commande svc_send reply. A quoi sert chacun des arguments ?

Q6 : comment les paramètres des services (ADD par exemple) sont-ils codés ?

Q7 : modifier le code de l'application cliente pour faire plusieurs appels vers le serveur.

Q8 : maintenant inserez une temporisation dans le traitement des opérations de la calculette. Pour cela, utilisez les commandes Unix de 'type sleep' (voir les documents systèmes en ligne. Lancez plusieurs clients dans plusieurs terminaux. Que se passe t il ?

(vous pouvez faire varier le temps de temporisation)

Partie 3 : Transformation vers une calculatrice plus classique

On s'intéresse désormais à la définition d'une calculatrice plus classique avec une mémoire mais qui admet 2 paramètres. Que faut il modifier ?

Reprenez les différentes étapes de génération (rpcgen … et exécution du make) telles que présentées précédemment.

Q7 : quelles sont les différences dans la spécification ? Comment ce nouveau type de données est-il gérer ? y a-t-il de nouveaux fichiers ?

Q8 : précisez enfin comment les erreurs sont gérées par ce type de RPC. Regardez le code et précisez les différentes erreurs qui sont mentionnées et référencées. Comment une applictaion peut elle les traiter ?

Partie 4 : vers un système tolérant aux pannes / fautes (à faire en autonomie et en binome)

Nous avons vu en cours que le traitement des erreurs est important dans les systèmes distribués au vu du nombre d'événements qui peuvent se dérouler : perte de message, perte de réponse, crash du serveur. Il s'agit donc dans cette partie du TP d'explorer les mécanismes de gestion et de traitement des erreurs.

- Quels sont les mécanismes proposés par les RPC pour essayer de relancer une requête du client sans réponse ? (pensez ici au timeout)
- Quels sont les mécanismes dont nous disposons pour détecter une panne d'une application serveur ?
- Lisez les documentations en ligne sur les RPC

- Écrire le code de l'application « calculatrice » des parties précédentes pour insérer ces mécanismes de détection de pannes.
- Quels sont les scénarios de test que vous pouvez réaliser dans ce TP ? exécutez les et analysez les résultats.
- Question bonus : quelles sont les différences entre gRPC (de Google) et les RPC (initiaux de Sun). Transformez le code de la calculatrice pour une execution avec protobuf et gRPC.

Partie 5 : Application du dictionnaire (pour les fanas de C !!!)

L'application d'origine (non répartie) servant de support à ce travail a été choisie pour sa simplicité. Elle serait très certainement perfectible, surtout du point de vue de l'interface utilisateur, mais son amélioration est totalement en dehors de l'objectif de ce projet.

Le fichier source, en langage C, du dictionnaire est contenu dans le fichier `dict.c`. Examiner et exécuter ce programme afin de bien comprendre ses fonctionnalités et son mode opératoire. Sa compilation peut être réalisée à l'aide de la simple commande :

```
gcc dict.c -o dict
```

- tester le code et regarder les différentes sections présentes
- quelles sont les structures de données utilisées ? quelles sont les structures de contrôle utilisées ?

On va définir l'interface du serveur et la programmer (fichier rdict.x)

Utiliser ensuite la commande comme dans la première partie
`rpcgen rdict.x`

À la suite de cette opération, les fichiers `rdict.h`, `rdict_xdr.c`, `rdict_clnt.c` et `rdict_svc.c` apparaissent. Rien n'est à modifier dans ces fichiers.

Générer ensuite les différents fichiers supports et de tests :

```
rpcgen -Sm rdict.x > Makefile
rpcgen -Sc rdict.x > rdict_clnt_main.c
rpcgen -Ss rdict.x > rdict_svc_proc.c
```

Compléter ensuite le fichier Makefile afin qu'il référence les fichiers utilisés dans ce projet en modifiant les deux lignes suivantes à l'aide des caractères en italiques :

```
SOURCES_CLNT.c = rdict_clnt_main.c
...
SOURCES_SVC.c = rdict_svc_proc.c
```

Les deux lignes :
```
CLIENT = rdict_client
SERVER = rdict_server
```

donnent les noms respectifs des exécutables client et serveur générés à l'aide de l'utilitaire `make`. Ces noms peuvent être modifiés à volonté, par exemple, et respectivement, en `rdict` et `rdictd`.

A partir du code du programme séquentiel, déplacer les différentes sections concernées dans les parties liées au client et au serveur. Testez le code !

Nous avons donc pu partir du code initial séquentiel écrit en C , l'adapter (avec l'écriture d'une interface, et un déplacement puis adaptation du code (les fameux pointeurs) et exécuter le code obtenu sur plusieurs machines du réseau de l'école…. En cela les RPC marquent une réelle rupture avec les sockets (et ce que nous avions programmé dans les TPs précédents).