TP ONC RPC

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.

Attention au copier / coller lorsque vous copiez à partir du pdf.... il y a parfois des caractères parasites!

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.

Q1 : Quels sont les services qui sont définis dans ce fichier ?

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 ?

Récupérer les fichiers calc_clnt_main.bon et calc_svc_proc.bon : comparer les avec la commande Unix "meld" (regardez en particulier le code qui est modifié entre les 2 versions), puis les renommer respectivement en calc_clnt_main.c et calc_svc_proc.c.
Examiner le code et comment le serveur est il mis en œuvre.

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 = *calcl_svc_proc.c*

Compilez le tout avec la commande **make** et lancez dans deux terminaux un exécutable de type client et l autre serveur. 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 ?

Transformation vers une calculatrice de nombres complexes.

On s'intéresse désormais à la même calculatrice qui gère cette fois ci non plus des entiers mais des complexes. On fournit pour cela une nouvelle interface calccomplexe.x

Reprenez les différentes étapes de génération telles que présentées précédemment.

Q7 : quelles sont les différences dans la spécification ? Comment ce nouveau type de données est-il gérer ? où et comment ?

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 2 : Application du dictionnaire

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. L'utilisation d'un programme serveur utilisant RPC implique l'enregistrement de ce dernier auprès du RPC portmaper de la machine sur laquelle il fonctionne. Afin d'éviter toute interaction entre les différents groupes de travail, il est important de suivre les conseils suivants :

- travailler exclusivement sur une station de travail personnelle et, en aucun cas, sur un serveur partagé ;
- arrêter le processus serveur du dictionnaire réparti avant d'en lancer un nouveau (après modification et recompilation, par exemple) ;

2.1 - Application non répartie

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 ?

2.1 - Application répartie

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 !

Seule la section suivante est à inclure dans votre rapport :

Modifiez le code du client afin que ce dernier garde les informations recherchées les plus récentes dans un cache (et cherche le cache avant de faire un appel distant). Donnez le code. Quelle est la complexité de cette extension ? quel est le gain lorsque l'information recherchée peut être trouvée localement ? quels sont les inconvénients d'une telle approche avec un cache ?