TD 11: Les RPC Unix

1ere Partie

Nous allons utiliser une solution basée sur les RPC qui fournit un dictionnaire accessible à distance. La version classique qui repose sur l'utilisation du portmapper ne fonctionne pas pour des raisons discutables de sécurité. Nous allons proposer une version qui n'utilise pas le portmap et illustre qu'il est inutile voir difficile d'interdire le lancement de services RPC sur une machine sauf à empêcher les connexions entrantes.

a. Utilisez la commande rpcinfo -p pour vérifier que le portmapper fonctionne et décrire les services RPC présents sur la machine.

b. Vérifier avec netstat -alnp | grep 111 que le processus portmap tourne sur la machine.

c. Compilez le code au moyen de make. Vérifiez que l'on a bien deux binaires exécutables l'un correspondant à un client et l'autre à un serveur, quels sont leurs noms ?

d. Exécutez le serveur. Vérifiez qu'il n'arrive pas à enregistrer le programme rpc auprès du portmapper pour les raisons de sécurité.

e. Modifiez le source rdict_svc.c pour que le serveur ne s'enregistre plus auprès du portmapper. Pour cela commentez pmap_unset et trouver la valeur du dernier paramètre de svc_register pour que le serveur ne s'enregistre pas.

f. Compilez et testez à nouveau le serveur.

g. Trouvez quel est le numéro de port utilisé par ce serveur via netstat -tlp

h. Vous allez devoir aussi modifier le client rpc rdict.c pour qu'il ne contacte pas le portmapper. Pour cela vous supprimez la ligne clnt_create

et vous allez la remplacer par le code fourni qui est à compléter afin de calculer une extrémité internet serveur au moyen de deux informations (nom du serveur, port du serveur) passées en paramètre du programme client.

i. Compilez et exécutez votre client avec les bons paramètres. Utilisez la commande 'I' pour initialiser le dictionnaire, 'i mot' pour insérer un mot, 'l mot' pour chercher ce mot dans le dictionnaire.

j. Commentez les paramètres de l'appel clnttcp_create pour justifier que cela permet bien de récupérer un numéro de descripteur réseau de type TCP afin de joindre le serveur RPC qui se nomme (argv[1]:RDICTPROG:RDICTVERS).

2ème Partie

Elle vise à faire comprendre que tout repose sur le fichier rdict.x de spécification du programme (ensemble de procédures). Cette spécification est partagée par le client et le serveur RPC. Nous verrons que le code à écrire pour le serveur se limite au fichier DictProcedures.hc et pour le client à inclure quelques fichiers générés automatiquement et à appeler la fonction clnt_create pour contacter le portmapeur ou dans sa version manuelle à créer soit même la "connexion" avec clnttcp_create.

a. Vérifiez via le fichier rdict.x que le client et le serveur partage bien la spécification d'un programme RDICPROG:RDICTVERS comprenant un ensemble de 4 procédures. Listez ces procédures accessibles à distance.

b. Vérifiez sur le code du client qu'il appelle bien une procédure initdict() lorsque l'utilisateur saisie le caractère 'I'.

c. Testez cette commande et commentez ce que fait le serveur.

d. Vérifiez que le serveur exécute bien la procédure initdict lorsqu'il reçoit la commande I en modifiant la trace de la fonction initdict.

e. Donnez les lignes de code qui ont été ajoutées pour que le programme se comporte comme un client RPC. Commentez la complexité par rapport à l'écriture manuelle d'un client utilisant directement l'API socket.

f. Vérifiez que le code à écrire pour le serveur se limite bien au fichier DictProcedures.hc. Que contient comme type de code ce fichier ? Trouve-t-on dans ce fichier des aspects réseau ?

g. Commentez la complexité par rapport à l'écriture d'un serveur utilisant directement l'API socket. Donnez des avantages et des inconvénients à cette approche.

Vous allez modifier la spécification du programme RPC pour autoriser un paramètre à la fonction initdict en définissant une structure DictName qui contient une longueur et une donnée (le nom du dictionnaire). Recompilez et corrigez les erreurs.