Examen de « Programmation Réseau »
Licence Informatique – Université Paris Diderot

Jean-Baptiste Yunès

17 Mai 2013

De nombreuses normes décrivent des mécanismes divers et variés de « calcul » à distance. Leur fonctionnement repose généralement sur trois entités : un annuaire, un serveur et un client.
L'annuaire recense l'ensemble des services de calcul qui ont bien voulu s'y déclarer.
Le client qui souhaite effectuer un appel à une fonction distante, interroge l'annuaire pour en obtenir les informations nécessaires à la réalisation de l'appel.
Le serveur qui désire offrir un service de calcul distant, se déclare auprès de l'annuaire en lui transmettant les informations utiles pour qu'un client puisse se connecter à lui, puis passe dans un mode passif dans lequel il attend des connexions entrantes afin de réaliser le service et d'en renvoyer la réponse. Un serveur peut attendre plusieurs connexions entrantes (une par fonction « exposée »).
Bien entendu pour garantir l'interopérabilité, il est nécessaire de fixer le format d'échange des données.
Pour répondre aux questions de l'examen vous devez choisir un langage parmi C ou Java.

Ordinateur, téléphones, etc sont interdits. La documentation papier est autorisée (sauf les feuilles du voisin).

1 Format d'échange de données

Une chaîne de caractère est codée sous la forme d'un triplet :
– le caractère `S`,
– longueur l (représentée sur un octet) dont la valeur est au plus 80
– et les l caractères de la chaîne (chacun 8 bits).
Le type correspondant s'appelera `netString`.

Un entier (toujours 32 bits) est représenté comme un couple :
– le caractère `I`,
– suivi par la représentation big-endian de sa valeur.
Le type correspondant s'appellera `netInt`.

Un message, c'est-à-dire une enveloppe d'objets de l'un des types précédents, transitant sur le réseau sera un couple :
– longueur l (représentée sur deux octets en big-endian),
– suite de l octets.
Le type correspondant s'appellera `netMsg`.

1. décrire les types dans le langage que vous avez choisi
2. écrire les fonctions `stringToNetString` et `netStringToString`.
3. écrire les fonctions `intToNetInt` et `netIntToInt`.
4. écrire les fonctions `addIntToNetMsg` et `addStringToNetMsg` permettant d'ajouter à un message existant et en fin de celui-ci un entier ou une chaîne de caractères passé(e) en paramètre. Attention le type

passé en paramètre est un type du langage, il devra donc devra être correctement transformé pour être ajouté.

À l'aide de ces fonctions, on va écrire des fonctions permettant de manipuler des messages.
D'abord des messages d'enregistrement de service. Ces messages sont constitués dans l'ordre, d'un entier de valeur 1 (pour représenté la volonté de s'enregistrer dans l'annuaire), d'une chaîne de caractère constituant le nom de la fonction distante qui pourra être appelée, d'un entier qui sera le nombre de paramètres que la fonction distante attend, d'un entier qui sera l'adresse de la machine sur laquelle le service de calcul est disponible, d'un entier qui sera le numéro de port auquel il faudra s'adresser pour obtenir le calcul.

5. écrire une fonction `infoServiceToNetMsg(netMsg,name,params,address,port)` permettant de fabriquer un message d'enregistrement et une fonction `netMsgToInfoService(netMsg, name, params, address, port)` permettant d'extraire d'un message les informations correspondantes. On précisera le type des arguments, etc

Ensuite des messages de consultation. Ces messages sont constitués dans l'ordre, d'un entier de valeur 2 (pour représenté la volonté de consulter l'annuaire), d'une chaîne de caractère constituant le nom de la fonction distante qui pourra être appelée, d'un entier qui sera le nombre de paramètres que la fonction distante attendra.

6. écrire une fonction `infoToNetMsg(netMsg,name,params,address,port)` permettant de fabriquer un message de consultation et une fonction `netMsgToInfo(netMsg,name, params, address, port)` permettant d'extraire d'un message de consultation les informations correspondantes. On précisera le type des arguments, etc

Et pour finir (il existe d'autres messages mais de structure relativement simple) des messages de réalisation d'appel. Un tel message contient une chaîne de caractère qui est le nom de la fonction, un entier n qui est le nombre de paramètres passés et n entiers qui représenteront les valeurs des paramètres.

7. écrire une fonction `callToNetMsg(netMsg,name,function,n,params,address,port)` permettant de fabriquer un message d'appel et une fonction `netMsgToInfo(netMsg, name, function, n, params, address, port)` permettant d'extraire d'un message de d'appel les informations correspondantes. On précisera le type des arguments, etc
8. écrire une fonction `writeNetMsg` permettant d'écrire un message dans un canal d'écriture, et une fonction
   `readNetMsg` permettant d'extraire un message depuis un canal de lecture.

2 Annuaire

L'annuaire est un service écoutant sur le port tcp `PPB` (Port des Pages Blanches), une constante définie dans votre langage dont la valeur importe peu. Il écoute en entrée sur son port différents messages.

9. écrire le code principal de l'annuaire permettant d'ouvrir une connexion de type serveur, d'attendre un message entrant, de l'extraire, de faire l'appel à un traitement (fonction `parseNetMsg`) puis de se remettre en attente. Inutile d'utiliser du multithreading...

Si l'enregistrement est possible, l'annuaire renverra un message constitué de l'entier 1 sinon 0 à celui qui l'a contacté.

10. écrire une fonction `doSubscribe` qui permet étant donné un message reçu au format précédent d'effectuer toutes les opérations nécessaires au bon fonctionnement futur du service.

Si la consultation aboutie le message renvoyé sera constitué d'un entier de valeur 1, d'un entier qui sera l'adresse de la machine sur laquelle le service de calcul est disponible, d'un entier qui sera le numéro de port auquel il faudra s'adresser pour obtenir le calcul. Si le service n'existe pas ou en cas d'autres problèmes l'annuaire renverra un message constitué de l'entier 0 .

11. quels cas d'erreurs sont possibles à la réception d'un tel message ?
12. quelles sont les conditions qui peuvent faire qu'une requête de ce type ne peut être réalisée ?
13. écrire une fonction `doSearch` qui permet étant donné un message reçu au format précédent d'effectuer toutes les opérations nécessaires au bon fonctionnement futur du service.

3 Serveur

Un serveur qui désire offrir un service de calcul doit établir un service de connexion sur lequel les clients pourront se connecter afin de réaliser des appels distants, puis d'enregistrer ce service auprès de l'annuaire. La réponse à un appel est un message constitué d'un entier qui s'il vaut 0 signifie que l'appel a échoué et 1 si l'appel a réussi. Si l'appel réussi, un autre entier représentant la valeur de la fonction est intégré au message.

14. écrire une fonction `createService` permettant d'initier la connexion d'un service de calcul
15. écrire une fonction `registerService` permettant d'enregistrer le service correspondant auprès de l'annuaire
16. écrire une fonction `wait` permettant au serveur de se mettre en attente d'appels entrants.
17. que faudrait-il faire de particulier dans ces différentes fonctions si on voulait permettre à ce que sur un même numéro de port, plusieurs fonctions puissent être appelées ? (en considérent que les fonctions peuvent être longues à être exécutées).

4 Client

Pour qu'un client puisse appeler une fonction distante, il lui est nécessaire d'obtenir les informations de connexions correspondantes. Pour cela, il lui faut d'abord interroger un annuaire (celui-ci est une machine connue dont l'adresse est contenue dans une chaîne de caractères constante `MPB` et le numéro de port `PPB`). Il lui faut ensuite établir une connexion vers la machine et le port indiqués dans la réponse de l'annuaire et envoyer les données demandées (un certain nombre d'entiers). En réponse il recevra l'entier résultat.

18. écrire une fonction `call(function,n,params,result` permettant de réaliser un appel à une `function` distante de nom donné prenant autant de paramètres que le tableau de `n`, entiers `params` l'indique et qui renvoie un entier en `résultat`. Attention la fonction `call` renvoie aussi un entier lequel doit valoir 1 si l'appel c'est correctement passé, et 0 sinon. Exemple (en C) :

```
int array[] = { 12,23 }; int result;
int r = call("mafonction",2,intarray,&result);
printf("%d\n",result);
```