INF8480

Module 2 : RPC

2.1 Un service de vote permet de voter (candidat et numéro d'électeur), et d'obtenir le résultat (nom du candidat et nombre de votes). Donnez le prototype de ces fonctions. Identifiez les paramètres d'entrée et de sortie.

```
void Vote(char *nom, int identificateur);
void Resultat(char **nom, int *nombre);
```

La première fonction a deux paramètres d'entrée alors que la seconde a deux paramètres de sortie.

2.2 Les électeurs veulent être certains que leur vote soit pris en considération. Quelle sémantique peut les satisfaire?

La sémantique peut-être ne suffit pas. Par contre au moins une fois est acceptable dans la mesure où le numéro d'électeur sert déjà à filtrer les répétitions.

2.3 Un réseau avec paquets qui peuvent être perdus est utilisé pour obtenir une sémantique au moins une fois. L'implantation peut ou non utiliser une hypothèse de délai maximal. Quel est l'effet sur l'implantation?

En retransmettant passé un délai maximal, ceci élimine en théorie la possibilité de recevoir deux réponses, une tardive et une pour la retransmission.

2.4 Comment peut-on générer un proxy pour une méthode comme Vote dans un langage comme C++ qui ne supporte pas la réflexion?

Le code pour le proxy est généré à partir du IDL. Le proxy contient une référence réseau vers l'objet réel et une méthode Vote qui envoie au processus de la référence réseau: le numéro d'interface, le numéro de méthode, l'identificateur de l'objet, le nom, et le numéro d'électeur. La procédure d'envoi identifie le type de l'ordinateur (gros/petit boutien) et retransmet le tout jusqu`à obtention d'un accusé de réception.

2.5 Un client prend 5ms pour calculer les arguments d'une requête RPC, le serveur prend 10ms pour traiter la requête. Chaque envoi ou réception de paquet demande 0.5ms au système d'exploitation et le temps d'envoi d'un message sur le réseau est de 3ms. L'encodage et le décodage des arguments prend 0.5ms par message. Quel est le temps pour effectuer deux requêtes si le client est séquentiel? Si le client est multi-fils mais le serveur demeure séquentiel?

calculer args + encoder + envoi + transmission + réception + décodage + traitement + encoder + envoi + trasmission + réception + décodage =
5 + .5 + .5 + 3 + .5 + .5 + 10 + .5 + .5 + 3 + .5 + .5 = 25ms,

soit 50ms pour deux requêtes séquentielles.

En multi-fils, le client commence sa seconde requête après l'envoi à 6ms. Sa seconde requête est envoyée après un autre 6ms, soit à 12ms. Le serveur reçoit la première requête à 9ms, la traite et la renvoie à 21ms. La seconde requête est déjà là et peut être traitée à ce moment. La seconde réponse est envoyée à 33ms, et parvient au client à 36ms, pour un total de 37ms.

2.6 Pour un système de ramasse-miettes réparti, les opérations AddRef et RemoveRef sont utilisées par les clients pour indiquer lorsqu'ils commencent ou cessent d'utiliser un objet réseau. Discutez-en les implications.

Si le message AddRef est perdu, l'objet pourrait être enlevé même si un client l'utilise toujours. Lorsqu'un message RemoveRef est perdu, un objet est maintenu même si aucun client ne l'utilise. Si le client termine abruptement sans faire de RemoveRef, l'effet est similaire. Il faut donc utiliser des accusés de réception et avoir une détection de clients qui terminent sans crier gare.

2.7 Faire une interface CORBA IDL pour un contenant de tâches qui se présentent sous la forme clé/valeur. Les opérations sont ajouter, enlever, lire.

```
typedef string Key;
typedef sequence<octet> Value;

interface TaskBag {
  readonly attribute long numberOfTasks;
  exception TaskBagException {long no; string reason;};
  void add(in Key key, in Value value) raises (TaskBagException);
  void remove(in Key key, out Value value) raises (TaskBagException);
  void read(in Key key, out Value value) raises (TaskBagException);
};
```

2.8 Pourquoi CORBA et les systèmes similaires n'offrent-ils pas de constructeurs

CORBA offre une interface et la notion d'instance n'est donc pas automatique. Le cas par défaut est une instance unique qui se confond avec l'interface. Au besoin il peut y avoir une méthode d'une interface qui exporte des objets réseau associés à une même interface.

2.9 Vous devez programmer un système d'appel de procédure à distance pour vérifier la disponibilité et le prix de certains items: void inventaire(in string id, out unsigned long quantity, out unsigned long price); où id est un identifiant typiquement de 10 caractères, la quantité environ 12 et le prix est un entier (en centièmes de dollars) aux environs de $100 (10000 cents). Vous avez le choix entre CORBA et gRPC. Quels sont les avantages de chacun. Si on ne tient pas compte des entêtes (e.g. HTTP2) mais seulement de l'encodage des arguments, CDR versus protobuf, lequel sera plus compact?

CORBA et gRPC sont relativement semblables. Les deux permettent d'interfacer facilement des programmes écrits en différents langages et ont été développés avec une attention particulière pour l'optimisation. CORBA est mieux établi et bien normalisé. gRPC est plus moderne et permet une plus grande flexibilité au niveau de la compatibilité (vers l'avant et vers l'arrière) car les champs sont auto-identifiés. CORBA prendra pour son encodage, outre le code identifiant la fonction, la longueur de la chaîne id arrondie au multiple de 4 octets supérieur plus 4 octets pour stocker la longueur, soit 16 octets pour un identifiant de 10 caractères, 4 octets pour la quantité et 4 octets pour le prix, pour un total de 24 octets. De son côté, protobuf demandera 1 octet pour identifier le champ et le type combinés pour id, 1 octet pour la longueur, 10 octets pour l'identifiant, 1 octet pour le champ et le type combinés et 1 octet varint pour la quantité, 1 octet pour le type et champ combinés et 2 octets pour le prix (10000 cents), pour un total de 17 octets. gRPC offre donc un compromis très intéressant de flexibilité (champs et types auto-identifiés), de simplicité d'encodage et de compacité.