UPMC
SORBONNE UNIVERSITÉS

4I404 - SRCS : Systèmes Répartis Client/Serveur
Examen du 15 mars 2017

Nom : **Prénom :**

Durée de l'épreuve : 2h - **Aucun document autorisé**
Téléphones portables, baladeurs et autres appareils électroniques doivent être éteints.
Le barème n'est donné qu'à titre indicatif, pour vous permettre de juger de la difficulté des questions.
Attention : l'énoncé est imprimé recto-verso sur **4 pages**.

Exercice 1 : Questions de cours (7 points)

Pour cet exercice répondez aux questions sur cette feuille

Question 1 – 0,5 point

Donnez la définition du polymorphisme.

Question 2 – 0,5 point

Pourquoi ne doit on pas utiliser les classes enveloppes Java pour faire de la synchronisation ?

Question 3 – 0,5 point

Quel patron de conception (design-pattern) utilise Java pour implémenter ses filtres de flux.

Question 4 – 0,5 point

Qu'est-ce qu'un appel de méthode *semi-synchrone* ?

Question 5 – 1,5 point

Donner à l'aide d'un schéma les étapes (et lignes de code) permettant d'envoyer un message en utilisant le protocol UDP.

Question 6 – 1,5 point

L'exécution de la commande `rpcgen -a truc.x` créer les fichiers suivants : `Makefile.truc`, `truc_clnt.c`, `truc_server.c`, `truc_client.c`, `truc.h` et `truc_svc.c`

Donnez le rôle de chacun de ces fichiers.

Question 7 – 1 point

On peut remarquer dans la question précédente qu'il n' y a pas eu génération du fichier `truc_xdr.c`. Pourquoi ?

Question 8 – 0,5 point

Pourquoi est-il nécessaire de redéfinir la fonction `svc_run` lorsqu'un client souhaite être rappelé sur un service RPC éphémère ?

Question 9 – 0,5 point

A quoi sert le *PortMapper* ?

Exercice 2 : Service de journalisation en Java (6 points)

Lors de la correction de cet exercice il sera tenu compte de la simplicité et de la lisibilité de la réponse. Chaque nouvelle classe devra être donnée dans son intégralité. Pour simplifier le code vous ne tiendrez pas compte de la gestion des exceptions et des imports. Enfin dans cet exercice vous n'utiliserez ni RMI, ni Corba

Dans cet exercice on veut implémenter un service de journalisation décentralisé en Java. Le principe de fonctionnement de ce service est le suivant :

- des demons tourneront sur les clients et enverront sur le serveur ce qu'ils lieront sur leurs entrées standards, après l'avoir affiché sur la console ;
- un serveur acceptera les connexions de clients (non connus à l'avance) et enregistra dans des fichiers disjoints les journaux qu'il recevra.

Question 1 – 3 points

Donnez l'implémentation d'un programme client qui affichera sur sa sortie standard les données lues sur son entrée après, les avoir envoyées au serveur au travers d'une socket TCP sur le port 8080. On supposera l'IP du serveur connue à l'avance et que ce dernier est actif.

Pour plus de généricité, votre solution devra se baser sur un filtre d'`OutputStream` capable de forwarder les données lues vers le serveur de journalisation.

Question 2 – 3 points

Donnez l'implémentation du serveur, sachant que ce dernier offrira à toutes les instances de votre client une connexion TCP sur le port 8080. Les données envoyées par chaque client devront être enregistrées dans un fichier dont le nom sera généré à la connexion à partir d'un compteur de connexion.

Votre solution devra assurer à chaque client un traitement équitable et une parallélisation des traitements des journaux reçus.

Exercice 3 : Vote électronique en RMI (8 points)

Lors de la correction de cet exercice il sera tenu compte de la simplicité et de la lisibilité de la réponse. Chaque nouvelle classe devra être donnée dans son intégralité. Pour simplifier le code vous ne tiendrez pas compte de la gestion des exceptions et des imports.

Dans cet exercice on veut implémenter en RMI un service de référendums électroniques centralisé. On considèrera qu'un vote est binaire : favorable ou défavorable. On supposera aussi que la taille du corp électoral est de 11 participants et que tous finiront par voter à chaque référendum. Les IP de ces participants ne sont pas connues à l'avance mais tous connaissent celle du serveur de référendums.

Chaque instance d'un vote sera identifiée par une chaine de caractères correspondant à la question posée.

Question 1 – 4 points

Dans un premier temps chaque participant devra envoyer son vote sur le serveur au moyen d'une API que vous définirez. Ce dernier ce contentera de comptabiliser les votes.

Donnez l'interface `Referendum` et implémentez de la façon la plus simple possible ce service RMI. Puis implémentez un serveur et un client permettant de tester votre service en votant à la question `"Ça marche ?"`. Le vote de chaque client sera défini par un paramètre de lancement du programme Java.

Question 2 – 4 points

On veut maintenant pouvoir offrir la possibilité de vérifier le vote de chaque participant en s'adressant directement à chacun d'eux. Le serveur ne compte plus les votes et se contente d'enregistrer les participants. Il doit donc maintenant offrir un service permettant de pouvoir connaitre les participants à un vote, en plus du service permettant de s'enregistrer.On doit aussi pouvoir interroger directement les participants sur leur vote.

Modifiez l'interface `Referendum` du serveur, proposez si nécessaire une autre interface et implémentez ces services. Vous modifierez en conséquence votre client et votre serveur.

Enfin, vous donnerez le code d'un programme Java `Scrutateur` qui :

1. récupèrera sur le serveur l'ensemble des participants
2. interrogera chaque participant directement pour connaitre son vote
3. affichera le résultat global du vote sur la console.

Pour simplifier le `Scrutateur` pourra rester bloqué si le vote n'est pas terminé, *i.e.* si tous les participants n'ont pas encore voté.

