Université des sciences et technologies de Lille (USTL) Juin 2008
Master mention Informatique – M1
Module : Conception d'applications réparties (CAR)

Examen – 2nde session

Conception d'applications réparties (CAR)

Tous documents papier autorisés. Téléphones et ordinateurs interdits.
Le barème est donné à titre indicatif. 3 heures.
Ce sujet comporte 4 pages.

1 Conception d'un protocole client/serveur (12 points)

On s'intéresse à la conception d'un protocole client/serveur de transfert de fichiers. Les serveurs gèrent un ensemble de fichiers. Les clients se connectent sur le serveur pour récupérer un fichier.

1.1 Le serveur est-il avec ou sans état permanent ? Si la réponse est avec état, quel est l'état ? Sinon, pourquoi n'y a-t-il pas d'état ? (1 point)

1.2 Plusieurs clients peuvent-ils transférer simultanément le même fichier ? Justifier. (1 point)

En cas d'interruption du transfert, suite à une panne réseau ou à une panne de serveur, on souhaite ne pas avoir à reprendre le transfert depuis le début lorsque le service redevient opérationnel (pour que les clients ne perdent pas le contenu des fichiers déjà transférés). On se base pour cela sur un mécanisme de « point de reprise majeur ». Tous les 100Ko transférés, le serveur envoie au client un message `PRep` (point de reprise majeur). Un point de reprise majeur est aussi envoyé au début, avant tout transfert de donné. Chaque point est numéroté à partir de 0. A la réception d'un message `PRep`, le client acquitte la réception en envoyant au serveur un message `PRepAck` pour indiquer au serveur qu'il a bien reçu le point de reprise.

1.3 Lors d'une demande de transfert de fichier par un client, quelle modification cette fonctionnalité entraîne-t-elle au niveau du protocole ? Que doit indiquer en plus le client ? (1 point)

1.4 Lors de l'envoi d'un message `PRep`, le serveur attend d'avoir reçu un acquittement (`PRepAck`) avant de continuer. Citer deux fonctionnalités qu'un tel mécanisme permet de réaliser. (1 point)

On introduit maintenant un nouveau mécanisme dit « point de reprise mineur ». Entre deux points de reprise majeurs, le serveur envoie un message `PRepMin` (point de reprise mineur) après 50 Ko transférés. Les points de reprise mineurs ne sont pas acquittés (i.e. les clients ne renvoient rien suite à leur réception).

Etude des messages échangés par le protocole client/serveur

On considère les **primitives** suivantes :
- `ouvrirCx(serveur)` primitive de demande d'ouverture de connexion vers `serveur`
- `attendreCx()` primitive d'attente bloquante et acceptation d'une demande de connexion

- `envoyer(message)` primitive d'envoi d'un message
  `message` peut prendre une des valeurs suivantes :
  - `TRANSFERT` *`nom i j`* demande de transfert du fichier *`nom`*
    message envoyé 1 fois à chaque demande d'un nouveau transfert
    si `j=0` demande de transfert à partir du point de reprise majeur `i`
    si `j=1` `demande` de transfert à partir du point de reprise mineur suivant le point de reprise majeur `i`
  - `DATA` *`i j`* envoi d'un bloc de données de au plus 50 Ko
    si `j=0` envoi à partir du point de reprise majeur `i`
    si `j=1` envoi à partir du point de reprise mineur suivant le point de reprise majeur `i`
  - `PRep` *`i`* envoi du point de reprise majeur numéro `i`
  - `PRepAck` envoi de l'acquittement d'un point de reprise majeur
  - `PRepMin` envoi du point de reprise mineur
  - `FIN` pour signaler la fin du transfert
- `recevoir()` primitive d'attente bloquante et réception d'un message
  (`message = recevoir()`)
- `fermerCx()` fermeture de connexion
- `nbBloc100K(fic)` retourne le nombre de bloc de au plus 100 Ko du fichier `fic` (3 pour un fichier de 225 Ko)

1.5 Dans le cas particulier où le fichier s'appelle `fic` et contient 225 Ko, indiquer sur un diagramme la séquence de **messages** échangés par le client et le serveur pour un transfert complet du fichier. Les messages sont ceux définis ci-dessus et uniquement ceux-là. On ne demande pas de faire apparaître les primitives sur ce diagramme. (2 points)

Implantation des services

On se replace dans le cas où la taille du fichier est quelconque. On considère les primitives suivantes :
- `client(fic,i,j)` exécutée par le client pour demander le transfert du fichier `fic` à partir de :
  si `j=0` du point de reprise majeur `i`
  si `j=1` du point de reprise mineur suivant le point de reprise majeur `i`
- `serveur()` exécutée par le serveur pour envoyer le fichier au client.

Dans un but de simplification, on suppose que :
- en cas de réception d'un message non attendu, les procédures sont interrompues et une erreur signalée à l'utilisateur,
- on ne s'intéresse pas à la façon dont le client écrit les données du fichier qu'il reçoit.

1.6 En utilisant les primitives de la question précédente, écrire le pseudo-code des primitives `client(fic,x,y)` et `serveur()`. (6 points)

2 CORBA – Gestion d'un parking (8 points)

On considère un parking dont on souhaite automatiser la gestion en l'administrant à distance avec un *middleware* CORBA. Le parking peut accueillir au maximum 50 véhicules. Une barrière gère les entrées et les sorties de véhicules : elle délivre un ticket à l'entrée d'un véhicule et elle contrôle le ticket fourni par l'automobiliste à la sortie. Un automate de paiement permet sur présentation du ticket, de régler le montant correspondant au temps de stationnement. L'automobiliste règle le montant du stationnement à l'automate avant de franchir la barrière pour sortir. La barrière, l'automate et le parking sont représentés chacun par un objet CORBA.

L'interface `BarriereItf` de la barrière fournit deux méthodes :
- `entrer` : correspond à l'entrée d'un véhicule dans le parking. Retourne vrai si le véhicule peut rentrer (si place libre), faux sinon (et dans ce cas le véhicule ne peut pas entrer). Cette méthode fournit en paramètre de sortie une structure `Ticket` comprenant un numéro de ticket de type entier et une heure d'entrée sous la forme d'une chaîne de caractères.
- `sortir` : correspond à la sortie d'un véhicule. Retourne vrai si l'automobiliste a réglé le montant du stationnement, faux sinon (et dans ce cas le véhicule ne peut pas sortir). Cette méthode prend en paramètre d'entrée une structure `Ticket`.

L'interface `AutomateItf` de l'automate fournit deux méthodes :
- `payer` : correspond au paiement en fonction de la durée de stationnement. Retourne vrai si le montant fourni par l'automobiliste est suffisant, faux sinon. Cette méthode prend en paramètre d'entrée une structure `Ticket` et un montant de type réel double. Cette méthode fournit également en sortie une valeur de type réel double représentant la monnaie à rendre à l'utilisateur.
- `cestpaye` : à partir d'une structure `Ticket` fournie en entrée, retourne vrai si le montant du ticket a été réglé, faux sinon.

L'interface `ParkingItf` du parking fournit trois méthodes :
- `nbPlacesLibres` : retourne un entier indiquant le nombre de places libres dans le parking.
- `entreeVehicule` : signale l'entrée d'un véhicule sur le parking. Ne prend aucun paramètre, ne retourne rien.
- `sortieVehicule` : signale la sortie d'un véhicule du parking. Ne prend aucun paramètre, ne retourne rien.

2.1 Définir la structure `Ticket` et les interfaces IDL correspondant à ces trois objets CORBA dans un module `ParkingPkg`. (2 points)

Arrivée d'un véhicule

2.2 Lorsqu'un automobiliste se présente à la `Barrière`, l'objet CORBA barrière interroge l'objet `Parking` avant de laisser entrer l'automobiliste. Pourquoi ? Quelle méthode invoque-t-il ? (0,5 point)

2.3 À l'issu de cela, il se peut que l'objet `Barrière` invoque la méthode `entreeVehicule` de l'objet `Parking`. Cette méthode n'a pas de paramètres et a pourtant un rôle essentiel. Lequel ? (0,5 point)

Sortie d'un véhicule

2.4 Décrire le scénario de la sortie d'un véhicule du parking : quel(s) objet(s) appelle(nt) quelle(s) méthode(s) de quel(s) autre(s) objet(s) et pourquoi ? En particulier, on s'attachera à fournir un scénario de fonctionnement permettant à l'objet `Barrière` d'autoriser ou non la sortie du véhicule. (1 point)

Implémentation du parking

2.5 Ecrire le code de la classe `ParkingImpl` implémentant l'interface `ParkingItf`. (1 point)

Ajout d'une barrière

2.6 On souhaite ajouter une deuxième barrière d'entrée/sortie au parking. Cela modifie-t-il les interfaces définies à la question 1 ? Si oui comment, si non pourquoi. (0,5 point)

2.7 La classe `ParkingImpl` de la question 5 est modifiée suite à l'ajout d'une barrière. Après avoir rappelé les méthodes invoquées lors de l'arrivée d'un véhicule, expliquez en quoi consiste cette modification (on ne vous demande pas de la programmer). (1 point)

2.8 Même question pour la sortie d'un véhicule. (1 point)

Evolution du parking

2.9 On souhaite maintenant connaître les heures d'entrée et de sortie des véhicules. Sans donner le code, expliquez comment vous pourriez faire cela. (0,5 point)