Programmation Client/Serveur en mode message

L'objectif du TD est d'étudier sur quelques exemples comment définir un protocole de communication entre des entités réparties.

1 La notion de protocole

Lorsque des entités réparties sur différentes machines interconnectées via un réseau souhaitent communiquer pour coopérer, elles ne disposent pas de mémoire commune ou de fichiers partagés. Elles doivent utiliser les moyens de communications offerts par le réseau. La communication se fait alors par échange de messages sur le réseau. Cet échange est dit asynchrone car il n'existe pas de référentiel de synchronisation entre les deux entités communicantes. L'entité A qui envoie un message vers l'entité B n'est pas bloquée après envoi du message. Elle peut continuer son exécution, si elle le souhaite. Par contre la réception d'un message est dite bloquante. L'entité B qui est en attente de réception de message est bloquée jusqu'à réception d'un message qu'elle traite ensuite. Ce mode de fonctionnement est dit mode de **communication asynchrone** ou encore **communication en mode message**.

Le protocole d'échange entre un navigateur Firefox ou Internet Explorer et un serveur Web est un protocole de ce type. Lorsqu'un navigateur souhaite obtenir une page HTML stockée sur un serveur, il envoie au serveur un message contenant une demande de document. Le serveur lui répond par un message contenant le document demandé.

Dans la suite de ce TD, nous considérons que le protocole de transport de messages utilisé est fiable : il n'y a pas de perte de messages, pas de doublons, les messages arrivent dans l'ordre d'émission, etc. Dans le monde Internet, la couche Transport TCP assure ces services.

Le protocole Transport n'est pas suffisant pour structurer un dialogue entre un client navigateur et un serveur web. Il faut définir un protocole applicatif. C'est le cas avec le protocole HTTP (HyperText Transport Protocol) qui sert de protocole applicatif entre navigateurs et serveurs Web. Un protocole applicatif est caractérisé de la manière suivante :

- le protocole de transport utilisé est TCP/IP pour HTTP
- les différents messages échangés entre les applications, par exemple les requêtes (GET, HEADER, PUT, POST, DELETE) et les réponses HTTP (401, 250,...)
- le format des messages, c'est-à-dire le type et la valeur des données constituant les messages. Par exemple, une requête HTTP de demande de document a la forme suivante :
  `GET /index.html Version/1.0`
- le séquencement dans lequel les messages seront échangés et la direction de l'échange.

Dans la suite, nous nous intéressons à la définition de protocoles applicatifs en étudiant sur quelques exemples quels sont les messages échangés, leur contenu et le séquencement de ceux-ci. Nous étudierons comment implanter ces protocoles au-dessus de protocoles de transport de messages comme UDP ou TCP.

2 Serveur calculette

En utilisant un protocole de transport fiable (TCP), on souhaite réaliser un serveur qui implante les fonctionnalités d'une calculatrice élémentaire fournissant quatre opérations (`+ - * /`). Des clients doivent donc pouvoir se connecter à distance et demander au serveur la réalisation d'une opération. Le serveur leur répond en fournissant le résultat ou un compte rendu d'erreur.

Question 1

Protocole applicatif

Définir un protocole applicatif (i.e. un ensemble de messages et leur enchaînement) aussi simple que possible implantant cette spécification.

Question 2

Cas d'erreur

Recenser les cas d'erreur pouvant se produire avec ce protocole applicatif et proposer un comportement à adopter lorsque ces erreurs surviennent (on ne s'intéresse ici ni aux erreurs de transport du style message perdu, ni aux pannes de machines).

Question 3

Codage

En utilisant les primitives fournies ci-dessous donner le pseudo-code du serveur et le pseudo-code d'un client demandant la réalisation de l'opération 8.6*1.75.

— `ouvrirCx(serveur)` : demande d'ouverture de connexion vers `serveur`
— `attendreCx()` : attente bloquante et acceptation d'une demande de connexion
— `envoyer(données)` : envoi de données
— `recevoir()` : attente bloquante et réception de données
— `fermerCx()` : fermeture de connexion

Question 4

Serveur

Le serveur est-il avec ou sans état ?

Question 5

Clients Multiples

On souhaite que plusieurs clients puissent se connecter simultanément sur le serveur. Cela pose-t-il un problème ?

Question 6

Suite d'opérations

On souhaite maintenant que les clients puissent enchaîner plusieurs demandes d'opérations au sein d'une même connexion. Proposer deux solutions pour cela (une en modifiant le protocole précédent et une sans modification du protocole).