TP N° 5 de Réseaux
Programmation d'une application client/serveur

Pascal Sicard

1 INTRODUCTION :

Vous connaissez les principes fondamentaux des réseaux informatiques mais sauriez vous programmer une application qui permette l'échange de données à travers un réseau ? C'est ce que nous vous proposons de réaliser dans ce TP.
Nous nous basons sur les protocoles de la famille TCP-IP (correspondant aux couches 4 et 3 de la norme OSI) et l'interface d'accès à ces protocoles s'appelle les Sockets. Ceux sont les mêmes primitives que vous avez utilisé avec ***SOCKLAB*** dans un précédent TP. Pour vous simplifier le travail (programmation de la gestion des erreurs, aides à la mise au point...) vous avez à votre disposition une boîte à outils permettant d'appeler simplement les primitives des sockets d'UNIX (voir la documentation jointe : ***Documentation technique sur les Sockets***).

2 Les applications proposées

Nous vous proposons de programmer une des applications suivantes :

1. **Transferts de fichiers**
   Plusieurs fichiers sont disponibles sur le disque dur d'un ordinateur. Des utilisateurs peuvent accéder à distance à ces fichiers. Il faut établir un protocole simple de dialogue entre le serveur et ses clients. Il est conseiller de se donner un cahier des charges le plus simple possible. Par exemple : le client envoie le nom du fichier demandé. Vous pouvez si le temps le permet de compléter l'application par d'autres fonctionnalités comme : le serveur envoie d'abord la liste des fichiers au client.

   Vous trouverez des détails sur les primitives de gestion de fichiers en C dans la documentation détaillée sur les sockets et les fichiers sur le Moodle de l'UFR.
2. **Dialogue au clavier**
   Application permettant un dialogue interactif du style de l'application ***talk*** d'Unix :

deux utilisateurs entrent des caractères au clavier sur des machines distantes, les caractères frappés sont transférés immédiatement et apparaissent sur l'écran de l'autre utilisateur.
A savoir la primitive *getchar* est bloquante tant que le caractère "entrée" n'est pas tapé.

Pour ceux qui veulent réaliser une application à l'interface ergonomique, vous avez un exemple de gestion de fenêtres dans le fichier : *EssaiCurses.c*. Attention la primitive *getch* ne peut être utiliser qu'avec l'utilisation des fenêtres de cette librairie.

3. **Le jeu du pendu**
   Nous vous proposons de programmer le fameux jeu du ***pendu*** dont voici un exemple de cahier des charges :
   - Le client se connecte au serveur.
   - Le client choisi un niveau de jeu (nombre maximal d'essais).
   - Le serveur lui annonce le nombre de lettres du mot mystère.
   - Le client envoie ensuite successivement des propositions de caractères. Le serveur retourne pour chaque essai une chaîne représentant l'état de découverte du mot mystère (par exemple des tirets montrent les caractères non trouvés) et le nombre d'essais restant à chaque tentative. On pourra s'amuser à figurer ce nombre d'essais par le dessin classique du pendu.
   - Le nombre d'essai restant diminue si la lettre saisie n'est pas contenue dans le mot mystère.
   - La partie s'arrête lorsque toutes les lettres du mot mystère ont été trouvées ou lorsque le nombre d'essai restant est nul.
4. **Autres idées à proposer à l'enseignant avant de se lancer (jeux simples comme le Mastermind ...)**.

3 Les algorithmes client/serveur

3.1 Généralités

Le concept client/serveur est devenu la méthode incontournable pour la communication au niveau application quand le protocole utilisé est de la famille TCP/IP. En effet, les protocoles de la famille TCP/IP ne fournissent aucun moyen de lancer un programme à distance grâce à l'envoi d'un message. De ce fait dans une communication entre deux parties, il faut forcément qu'il y en est une en attente permanente d'une requête éventuelle de l'autre.

Les applications sont donc classées en deux catégories :

- Les clients : ils prennent l'initiative du début de la communication (demande d'ouverture de connexion, envoi de requête, attente de réponse)

- Les serveurs : ils sont en permanence à l'écoute, en attente de demande de communications (attente de demande d'ouverture de connexion, de requête, émission de

réponse).

La procédure générale de dialogue entre un serveur et son ou ses clients est la suivante :

- Le **serveur** initialisé se place en écoute (attente de requête client).
- Le **client** émet une requête à destination du serveur.
  Il demande un service ou l'accès à une ressource.
- Le **serveur** renvoie une réponse au(x) client(s).
  Il rend le service ou octroie la ressource, éventuellement il engage même un dialogue assez conséquent avec son (ses) client(s).
- Le **serveur** se replace en écoute (attente de nouvelle requête client).

C'est toujours le client qui a l'initiative de l'échange !

3.2 Les différents types de serveurs

On peut classer les serveurs dans deux types distincts :
- **Serveur itératif** : les différentes requêtes sont traitées les unes après les autres par un seul processus.
- **Serveur parallèle** ou à **accès concurrents** : les différentes requêtes sont traitées en parallèle, chacune nécessitant la création d'un processus dédié à son traitement.

3.2.1 Serveur itératif

1. Les requêtes sont traitées séquentiellement par le même processus. Les requêtes qui arrivent pendant un traitement sont mémorisées par le processus « transport » (TCP ou UDP).
2. Il peut y avoir plusieurs clients qui dialoguent en même temps avec le serveur, les requêtes sont mémorisées dans une seule file d'attente et le serveur différenciera les requêtes au moment de leur traitement.
3. Ce type de serveur est facile de conception, mais il se peut que certaines requêtes dont le temps de traitement est très élevé créent un délai inacceptable pour le traitement de requêtes d'autres clients. On utilise dans ce cas un serveur dit « parallèle ».

3.2.2 Serveur parallèle (accès concurrent)

Chaque requête est traitée par un processus fils dédié créé à cet effet. Une requête "trop longue" ne pourra plus retarder d'une façon trop importante d'autres traitements.

3.3 Modes de communications

Pour chacun de ces deux types de serveurs, on peut écrire des applications dites orientées connexion ou pas suivant le protocole transport utilisé. Par définition, un serveur utilisant TCP est dit orienté « connexion » et un serveur utilisant UDP est dit orienté « sans connexion ». Suivant l'application à réaliser, le choix de l'un ou l'autre de ces protocoles est plus ou moins adéquat en fonction des qualités de services désirées.

4 Environnement et méthodes de travail

- Après avoir lu l'énoncé du TP et la documentation technique sur les Sockets, définissez un cahier des charges précis de votre application.

- Choisissez ensuite le type de serveur et le mode de communication adéquat à votre application. Justifiez vos choix.

- Ecrivez l'algorithme du serveur et du client de votre application. Ecrivez ensuite les programmes C correspondants.
  Vous pouvez vous servir des fichiers ***client.c*** et ***serveur.c*** qui sont à votre disposition.

- Un ***makefile*** permettant de compiler facilement le client et le serveur est aussi à votre disposition dans le placard électronique.

- Le fichier ***fon.c*** est nécessaire pour utiliser la boîte à outil au dessus des sockets.

- Vous pouvez utiliser **Socklab** afin de débugguer indépendamment le client et le serveur.

5 Compte-rendu

- Cahier des charges de l'application

- Solutions choisies plus leurs argumentations

- Description des protocoles utilisés dans votre application (Dialogues client/serveur)

- Programmes C fortement commentés - Démonstration et/ou jeux de tests

6 Pour ceux qui veulent aller plus loin

Essayez les applications ***talk*** et ***ftp***, expliquez en détail comment elles fonctionnent au niveau du réseau : protocole de niveau transport utilisé, connexions ouvertes, numéro de port... etc. Servez vous si nécessaire des outils de capture de trames que vous avez à votre disposition.

7 Références Techniques :

Manuels free-BSD : man socket, man curses

A disposition sur le moodle :

- Documentation résumée sur les Sockets
- Documentation C détaillée (socket, fichiers)