Skaïpeuh
Projet de « Programmation Réseau »

Licence Informatique

Université Paris Diderot

I. Briquel, M. Renault, A. Sangnier, J.-B. Yunès

2014

Introduction

Le but de ce projet est de programmer un système de **communication entre utilisateurs**.

Le système sera composé d'applications communiquantes en TCP et en UDP à l'aide d'un protocole applicatif (décrit dans la suite). Un groupe-projet réalisera une application communiquante qui devra impérativement communiquer avec celles d'autres groupes. Le protocole est conçu pour permettre l'interopérabilité.

L'idée est relativement simple. Un utilisateur souhaitant communiquer lancera l'une des applications disponibles. Celle-ci établira une **connexion** au système via un protocole applicatif par datagrammes (sur UDP). Cette connexion permettra essentiellement à l'utilisateur de se déclarer aux autres présents dans le système. De cette connection l'utilisateur obtiendra la liste des autres utilisateurs connectés (cette liste devra être maintenue à jour en temps-réél comme on le verra ensuite) et il pourra donc obtenir une communication privilégiée avec l'un de ceux-ci. Cette communication sera effectuée en mode connecté (sur TCP) et consistera principalement en l'échange de messages frappés au clavier.

Dans la suite le signe ␣ représentera un simple caractère d'espacement (ASCII 32). Pour toute description précise des formats à employer, reportez-vous à la fin du document.

Description détaillée

Le système

Comme on l'a déjà indiqué, avant toute communication entre utilisateurs, une phase d'entrée dans le système est nécessaire. Cette phase utilise uniquement des datagrammes UDP.

Tout d'abord il est nécessaire de préciser que toute entité du système fait partie d'un groupe de multidiffusion, le numéro IP du groupe est `224.5.6.7` et son port le `9876` (vous pouvez utiliser d'autres valeurs dans des phases de tests, mais le jour de la soutenance il est impératif que ces valeurs soient employées).

Lorsqu'un utilisateur souhaite entrer dans le système, l'application qu'il lance, s'y déclare par un message de multidiffusion (*multicast*) de la forme suivante `HLO`␣*user*␣*machine*␣*port* (pour la description du format des paramètres reportez-vous à la fin du document). Le paramètre *machine*

représente l'adresse de la machine sur laquelle l'application a été lancée, et *port* un numéro de port TCP sur lequel l'application recevra les connexions entrantes pour les communications privées. De la sorte, les autres applications seront prévenues qu'un nouvel utilisateur est présent avec lequel elles pourront tenter de communiquer.

À réception d'un message de type `HLO`, chaque application présente dans le système répond par un message `IAM`␣*user*␣*machine*␣*port*. De la sorte l'application entrante est prévenue de l'existence d'applications déjà présentes.

Lorsqu'un utilisateur souhaite se déconnecter du système, son application devra envoyer un message multidiffusé de la forme `BYE`␣*user*. Ainsi, les autres applications seront prévenues que l'utilisateur correspondant a quitté le système.

Il existe aussi un message de contrôle de la forme `RFH` que n'importe quelle application peut multidiffuser à n'importe quel moment et dont l'effet pour chaque receveur est de renvoyer obligatoirement un message de type `IAM`. Ces échanges auront pour but de faire en sorte que chaque entité présente puisse remettre à jour sa liste courante d'applications présentes si une entité le considère comme nécessaire.

Un autre message de contrôle permet de bannir un utilisateur du système. Le message aura la forme `BAN`␣*user*. Attention : pour qu'un utilisateur soit réélement banni, il faut qu'au moins deux messages de bannissement aient été envoyés par deux applications différentes. Le bannissement consiste pour chaque application à faire comme si l'utilisateur avait envoyé un message de type `BYE`.

Attention : les messages du protocole échangés en UDP doivent être envoyés individuellement chacun dans son propre paquet UDP.

La communication privée

Une communication privée est un simple échange de messages de la forme : `MSG`␣*length*␣*seqchars* envoyés dans le canal TCP. Cet échange commence dès la connexion sur le numéro de port TCP choisi au lancement de l'application et qui a été donné à l'entrée dans le système dans le message de type `HLO`. L'échange prend fin dès lors que l'une des deux parties envoie un message `CLO` sur le canal de communication. Après avoir envoyé ce message, la partie ferme proprement son canal de communication et le récepteur fait de même à la réception du message.

Attention : un utilisateur ne peut mener qu'une seule conversation à la fois...

Durant la communication l'une des deux parties peut souhaiter envoyer un fichier à l'autre. Pour se faire, ils s'échangent les messages suivant :

1. `FIL`␣*filelength*␣*filename*␣*port*
2. si le receveur accepte l'échange, il renvoie un simple message `ACK` et l'échange se produit tel que décrit dans la section suivante et en mode concurrent (l'échange de fichier ne doit pas perturber la communication ordinaire).
3. si le receveur refuse l'échange, il renvoie un simple message `NAK`.

L'échange de fichiers

Une fois l'échange accepté, il suffit alors à l'entité réceptrice du fichier d'établir en concurrence (*i.e. : via* un *thread* ou un processus) une communication initiée sur le port TCP indiqué dans le message initial, puis de réceptionner les données brutes envoyées. Nous insistons sur le fait que cette fonctionnalité ne doit pas bloquer le reste de l'application. Les fichiers seront **toujours** réceptionnés dans un répertoire particulier de l'utilisateur : le répertoire `downloads` de son répertoire privé (et

aucun autre répertoire). Si un fichier du même nom y existe déjà, ce dernier sera rebaptisé à l'aide d'un suffixe adéquat quelconque permettant de les distinguer). La communication prendra fin lorsque l'envoyeur aura clos le flux.

Réalisation

La réalisation se fera nécessairement en C ou en Java. Un groupe **d'au moins deux étudiants et d'au plus trois** (ni plus ni moins) devra réaliser une application conforme à la description.

Il est impératif que chaque groupe utilise le système de dépôt `git` offrant des capacités de versioning que l'UFR possède : `moule.informatique.univ-paris-diderot.fr:8080`. Vous pouvez vous y connecter en utilisant votre compte de l'UFR (ne pas utiliser de compte « extérieur »). Apprenez à utiliser `git`, ce point sera particulièrement apprécié ! Pensez aussi à inviter les enseignants (obligatoirement le responsable du cours !) dans le projet `git` que vous aurez créé et leur donner suffisamment les droits *master*.

Il est **impératif de respecter** scrupuleusement la spécification fournie dans le sujet ainsi que les formats de message (sauf erreur manifeste dans le sujet et qui sera corrigé au plus vite). Toute violation sera jugée très défavorablement !

Il vous sera, de plus, fourni un exécutable permettant de tester certaines fonctionnalités basiques. Un point très important est de **faire inter-opérer votre application avec celle d'un autre groupe** voire plusieurs, voire tous.

Attention si pour obtenir ce qui est demandé ci-dessus, la communication verbale entre groupes est non seulement autorisée mais encouragée il est **strictement interdit** d'échanger du code ; ceci serait considéré comme plagiat et par conséquent jugé sévèrement (ne comptez surtout pas sur une quelconque incompétence des examinateurs à détecter du plagiat). Les discussions doivent donc seulement porter sur le fonctionnement du protocole et son interprétation, ou les formats des messages ; il vaut donc mieux éviter de donner trop d'indications sur la façon de coder les fonctionnalités (le cours, internet, etc vous donneront toutes les indications nécessaires, **à condition d'éviter comme la peste** les sites comme `codeparcopiercoller.fr`, `programmesansrienfaire.com`, `laprogrammationproen5minut.es`).

La **notation finale** prendra en compte le fait que des groupes auront réussi à faire communiquer leurs applications entre elles. Il faudra donc penser à en faire la démonstration. D'une certaine manière, ceux qui collaborent dans les limites indiquées ci-dessus seront récompensés ; pour ceux qui décident de faire quelque chose dans leur coin et qui ressemblerait vaguement à ce qui est décrit ce sera l'inverse, la notation sera revue à la baisse. Un **programme qui s'exécute n'est pas suffisant** ! Vous **devez** écrire un programme qui se comporte comme indiqué ; et s'il ne communique pas avec le programme écrit par d'autres, c'est que vous n'avez pas compris ce qu'est la programmation réseau.

Nous mettrons en place un forum sur didel pour que vous puissiez communiquer entre vous et avec nous, par exemple pour faire part d'imprécisions dans le sujet, ou pour signaler que vous avez une application qui tourne et ainsi donner l'opportunité à vos camarades de s'y connecter. **Dans tous les cas**, vous pouvez également contacter vos enseignants par mail ou lorsque vous les croisez à l'occasion.

Pour la soutenance, il sera **nécessaire de prévoir un mode de fonctionnement verbeux** dans lequel suffisamment d'informations seront disponibles à l'écran pour comprendre ce qui se passe (affiche des adresses utilisés, ports, états des listes internes, etc).

Il n'est pas demandé d'utiliser une interface graphique ; il est même recommandé de s'abstenir d'en faire une (quoique vous en pensiez). Cela ne vous apportera rien, vous fera perdre du temps et n'impressionnera en rien les examinateurs ; et surtout **ce n'est absolument pas dans le programme de cet enseignement** par conséquent, nous insistons : **pas d'interface graphique**.

Prenez garde à ne pas vous fier à la rumeur, les seules sources d'information fiables sont vos enseignants ! Au moindre doute, demandez-leur !

Formats

user : une suite de 8 caractères ASCII. Si le nom de l'utilisateur est de longueur inférieure à 8, le caractère ASCII `32` sera employé comme caractère de bourrage. Ainsi pour l'utilisateur « yunes » la suite `yunes␣␣␣` devra être employée.

machine : représente le codage de l'adresse IPv4 d'une machine. Ce codage est sur 15 caractères : 4 groupes de trois chiffres décimaux séparés par le caractère `.` (point). Le caractère `0` est employé comme caractère de bourrage. Ainsi l'adresse IPv4 192.168.1.0 sera représentée par la suite de caractères `192.168.001.000`.

port : représente le codage d'un numéro d'un numéro de port. Ce codage est sur 5 caractères décimaux avec `0` comme caractère de bourrage. Ainsi le numéro de port 345 sera représenté par la suite de caractères `00345`.

length : représente une longueur comprise entre 1 et 500. La valeur est codée sur trois chiffres en utilisant le caractère `0` comme bourrage. Ainsi la longueur 23 sera codée par la suite de caractères `023`.

seqchars : est une séquence de caractères. Celle-ci n'a pas de longueur en tant que telle, si une longueur doit être précisée le protocole fournit cette information par ailleurs. Les caractères sont de simples caractères ASCII 8-bits de l'encodage latin-1. Attention : il n'y a **pas** de caractère de fin ou quelque chose de cet ordre.

filelength : représente une longueur (éventuellement nulle). La valeur est codée sur autant de chiffres décimaux que voulu. La longueur 234876 pourra être codée par la suite de caractères `234876`, mais pourra très bien l'être aussi par la suite `0000000234876`.

filename : représentera un nom de fichier encodé par une suite de 40 caractères ; suite uniquement constituée de caractères alphabétiques non accentuées (majuscules ou minuscules), chiffres et `.` (point) et rien d'autre (pas d'espaces, souligné, tiret, etc). Le caractère ␣ devra être employé comme caractère de bourrage. Ainsi le fichier « superfichier.txt » sera encodé sous la forme
`superfichier.txt␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣`.