L3 Informatique :
HLIN503 Réseaux

Il est nécessaire d'avoir aussi la feuille de TD, certains exercices seront à rendre selon les modalités précisées par votre chargé de TD/TP

1 TP1 et TP2 : Connexion à distance

1.1 SSH - secure shell

1.1.1 Un début

Comme son nom l'indique `ssh` (*secure shell*) est un outil de connexion *sécurisé*. Première caractéristique : il ne transporte pas le mot de passe «en clair». Son utilisation la plus simple est :

`ssh nomHôte`

Par exemple, l'utilisateur `fasol`, connecté sur l'hôte `chuila` veut se connecter à l'hôte `charive` ; il va lancer dans une fenêtre (`xterm`) la commande :

`ssh charive`

Si c'est la première demande de connexion par `ssh` que `fasol` effectue sur `charive`, il aura une réponse ressemblant à :

```
The authenticity of host 'charive' can't be established.
RSA key fingerprint is ef:07:d1:1c:4d:03:6b:11:27:42:68:0f:77:a9:58:10.
Are you sure you want to continue connecting (yes/no)?
```

Après une réponse positive (`yes`), le mot de passe de fasol sur `charive` lui sera demandé et la connexion établie si ce mot de passe est correct. A partir de là, noter que toutes les commandes demandées dans cette fenêtre seront exécutées sur `charive`. En quelque sorte, seul l'affichage sur `stdout` est exécuté sur l'hôte de première connexion, ici `chuila`.

Par la suite, toute demande de connexion sur `charive` va requérir le mot de passe de fasol sur cet hôte.

Question laissée en suspens pour l'instant : que va-t-il se passer pour les affichages X11 ?

Répertoire et fichiers créés : sur `chuila` un répertoire `.ssh` contenant les fichiers `known_hosts` et (pas toujours) `random_seed` sont tous créés dans le répertoire d'accueil du demandeur. En particulier, le fichier `known_hosts` va contenir une ligne par hôte vers lequel on voudra se connecter par `ssh`. Ligne longue, contenant une *clef*. Pour l'instant, il suffit d'admettre qu'une *clef* est une suite alphanumérique, non nécessairement secrète.

Remarque : Sur le site `info-ufr.univ-montp2.fr` il est impossible de se rendre compte que c'est sur l'hôte de départ que les répertoires et fichiers sont créés, car l'ensemble de tous les répertoires utilisateurs sont distribués sur tous les hôtes. Du coup, toute création de répertoire ou de fichier sur un hôte quelconque est visible sur tous les hôtes. Il est impossible de savoir sur quel hôte la création ou modification s'est faite.

1.1.2 Premières questions

Une syntaxe un peu plus complète de `ssh` est :

```
ssh [-l login_name] hostname | user@hostname [command]
```

Elle permet d'exécuter des commandes à distance et aussi de demander une connexion en changeant d'identité.

1. Quel comportement (mot de passe en particulier) et quel résultat peut-on prévoir en lançant la commande `ssh charive ps -ef` sur `chuila` ?
   Fasol peut aussi demander `ssh -l rémi charive` ou encore `ssh rémi@charive`. Dans ce cas,
2. Que peut-on prévoir comme comportement et résultat ?
3. Que se passe-t-il s'il lance la commande
   `ssh -l rémi plusloin.tataouine.lesbains` ?
4. Que se passe-t-il si on fait `ssh charive` sur `chuila` puis dans la même fenêtre `ssh chuila` ? Indiquer le comportement et le contenu des fichiers dans `.ssh` .

1.1.3 Distinguer cryptage et authentification

Quand on parle de chiffrement, on pense à un ensemble de techniques permettant de masquer des informations et de ne les rendre lisibles qu'aux personnes *habilitées*. Notre but ici n'est pas de faire un cours sur ces techniques. On admettra donc qu'on parle de *cryptage* en pensant au codage de l'information, la rendant illisible à toute personne ne possédant pas une *clef* (données spécifiques) permettant de déchiffrer l'information.

On parle d'*authentification* lorsqu'on veut être sûr de la source de l'information reçue ; en effet, la réception d'un message codé ne permet pas forcément de garantir que l'expéditeur est bien celui attendu.

Dans le peu traité ici, on admettra que le l'ensemble du chiffrement repose sur le cryptage dit *asymétrique*, c'est-à-dire reposant sur l'existence de deux clés, l'une dite privée, restant chez le codeur, l'autre publique, qui peut être transportée. La connaissance de cette dernière ne suffit pas pour décrypter, on s'en doute.

1.1.4 Se passer du mot de passe

On étudie de plus près le fonctionnement du cryptage avec deux clefs, l'une publique, l'autre privée ou secrète.

Soient deux entités A et B avec les clefs k_p (publique) et k_s (secrète) sur A , k'_p et k'_s sur B. A veut envoyer un message m à B de telle sorte que B seul puisse le décrypter. B fournit à A sa clef publique k'_p. A calcule $m' = k'_p(m)$ (m' est le message m crypté par k'_p). Le décryptage sur B se fait en calculant $m'' = k'_s(m')$ avec $m'' = m$. Ainsi, on peut transmettre sa clef publique à quelqu'un afin de crypter un message qui nous sera envoyé et que personne d'autre ne pourra décoder, car la clef secrète est nécessaire au décryptage.

Ce système est utilisé aussi pour l'authentification (signature) avec `ssh` mais cette fois-ci le demandeur va crypter son identité par la clef secrète : si m est son identité, c'est $m' = k_s(m)$ qui est transmis. k_p peut être transmis avec le message ou avant encore. Tout destinataire pourra vérifier que c'est bien lui en décryptant avec la clef publique transmise : il calcule $m'' = k_p(m')$ et obtient l'identité de l'expéditeur : il doit obtenir $m'' = m$.

Le principe de la connexion évitant de donner un mot de passe repose sur cette deuxième caractéristique. Pour une connexion de `chuila` vers `charive` il faudra :

1. générer deux clefs d'authentification, une publique et une secrète sur `chuila`,
2. transporter la clef publique vers `charive`,
3. ensuite, toute connexion dans le sens `chuila` vers `charive` s'effectuera sans mot de passe, à savoir que toute demande de connexion sera cryptée sur `chuila` avec la clef secrète et décryptée à l'arrivée avec la clef publique précédemment fournie.

Comment faire pratiquement ?

1. la génération des deux clefs se fait par `ssh-keygen -t rsa` (rsa étant une méthode classique de cryptage) sur `chuila` (en TP, ne PAS mettre de pass-phrase, c'est assez compliqué comme ça) ; le résultat est la création de deux fichiers toujours dans le répertoire `.ssh`, nommés (par défaut) `id_rsa` (clef secrète) et `id_rsa.pub` (clef publique) ;
2. le transport de la clef publique par tout moyen à votre convenance, sur le site `info-ufr.univ-montp2.fr` ça peut se faire par simple copie (mais pourquoi pas `scp`, voir plus loin) ; cette clef est à mettre dans un fichier nommé `authorized_keys` sur l'hôte destinataire ;
3. se connecter.

1.1.5 Questions

La génération des clefs d'authentification est propre à chaque utilisateur. On peut dire que ces clefs identifient un couple (hôte, utilisateur). Quelles sont les conséquences de la copie d'une clef publique `(rémi, chuila)` dans les autorisations (fichier `authorized_keys`) de fasol sur `charive` ?

Finalement, quelles sont les protections nécessaires sur tous ces fichiers ?

Exercice 1

La commande `ssh` va servir pour effectuer des connexions à distance. Voir à travers le manuel et la feuille de TD sa syntaxe si on ne s'en souvient pas. Faire d'abord `ssh` d'un hôte à un autre, dans sa forme la plus simple.

Ensuite, faire `ssh -v` (`v` pour *verbose*) avec les mêmes paramètres.

1. Noter ce qui se passe.
2. Quels sont les processus qui ont été mis en œuvre respectivement sur chaque machine pour réaliser cette connexion ?

3. Parmi tous les processus portant le même nom, noter où est le serveur (sshd) et le client ; peut-on identifier et distinguer ceux qui ont été engendrés par votre connexion ? Voir pour ceci les détails de la commande `ps` .
4. Peut-on se connecter ainsi à un site distant quelconque ?
5. Lancer `ssh` afin d'exécuter une commande à distance, `ps -ef` par exemple, sans connexion à la machine distante.

Constater qu'on ne peut pas lancer ainsi (selon la question 5 une commande qui génère une nouvelle fenêtre, par exemple votre éditeur préféré (xemacs par exemple).

Pour arriver à exécuter des applications avec transfert de l'affichage, il faut utiliser `ssh -X`, ou un peu mieux, `ssh -Y`. Voir le manuel à ce sujet. Constater ce qui se passe. Si on lance ainsi une application (par exemple le même éditeur préféré), où s'exécute l'application ? Indiquer comment vous vous y prenez pour répondre.

Essayer de mettre en place le nécessaire pour ne pas donner de mot de passe.

— Quels sont les fichiers créés ? Noter leur protection.
— Noter dans quel sens on peut ne pas donner le mot de passe.
— Penser à relancer toutes ces commandes avec l'option `ssh -v`.
— Comment faire pour essayer de se connecter sous le nom de votre collègue de travail ? Que faut-il mettre en place pour que cette connexion marche ? Revoir le TD à ce sujet.

2 Copie de fichiers

La syntaxe simplifiée est

```
scp [[user@]host1:]file1 [...] [[user@]host2:]file2
```

On constate donc qu'on peut être sur une une machine A et faire des copies entre deux autres machine B et C, même en ayant une identité différente sur ces machines.

1. `scp` utilise `ssh`. Analyser plusieurs situations pour les divers hôtes. Que peut-on prévoir comme demandes de mots de passe ? Expliquer son fonctionnement tel que cette syntaxe le suggère.
2. Quels sont les droits nécessaires à ce fonctionnement ?
3. Dans le cas du site `info-ufr.univ-montp2.fr` comment peut-on mettre en évidence ces copies ?

3 Transfert de fichiers

Dans tous les cas suivants, il est recommandé de déposer des fichiers dans le répertoire `/tmp` afin de pouvoir distinguer ce qui est sur chaque hôte. En effet, l'espace de travail courant est disponible sur tous les hôtes, alors que `/tmp` est propre à chaque hôte et permet de vérifier plus facilement si les résultats sont conformes à l'attente.

Exercice 2

sftp est une version plus moderne de ftp, outil simple et efficace de transfert de fichiers, comportant un cryptage et une authentification.

Une syntaxe simple est :
`sftp user@hôte`

1. Lancer cette commande sur un hôte local, puis faire `?` pour connaître les commandes possibles de sftp. Noter que certaines commandes ressemblent à celles du shell, mais d'autres non. Donc il est très utile dávoir une fenêtre ouverte sur le manuel de sftp.
2. Copiez chez vous un fichier venant de l'hôte distant (commande `get`. Est-il possible de récupérer plusieurs fichiers en une seule commande (voir `mget`) ?
3. Déposez un fichiers sur l'hôte distant (commande `put`).

Exercice 3

On peut copier des fichiers avec `scp`, d'un hôte vers un autre. Noter le comportement avec et sans mot de passe.

Pour diminuer les temps de transfert des fichiers, on utilise la compression. Plusieurs outils répondent aux problèmes de :

- compression de fichiers ; par exemple `gzip`.
  La syntaxe dans sa forme la plus simple est
  - `gzip nom_de_fichier` pour compresser ;
  - elle est `gzip -d nom_de_fichier` pour décompresser un fichier ;
- transformer une arborescence en un unique fichier ; par exemple `tar`, dont la syntaxe dans sa forme la plus simple est :
  - `tar cf nom_repertoire.tar nom_repertoire` pour créer un fichier *taré* (premier paramètre) à partir d'un répertoire donné (le deuxième paramètre),
  - `tar xf nom_repertoire.tar` pour extraire à nouveau le contenu.

Attention : `tar` ne détruit pas le fichier ou répertoire origine, donc il faut penser à le faire si on ne veut pas conserver les deux exemplaires ; par contre, `gzip` le fait.

Remarque : on peut avantageusement utiliser un *tube* entre `tar` et `gzip`, ce qui est une bonne façon d'archiver des répertoires. Réaliser un tel essai et comparer la taille des objets avant et après compression. Noter enfin que certaines versions de `tar` offrent un paramètre `z` permettant de compresser directement, sans avoir à passer par le tube.

Est-ce que l'outil `ftp` que vous utilisez permet de transférer des fichiers compressés sans erreurs ? Des (sous-)arbrescences ? Comment peut-on envisager de transférer des (sous-)arborescences, avec les outils que vous connaissez (y compris la messagerie) ?

Exercice 4

L'utilisation de `uuencode` devient obsolète ; Mais son fonctionnement permet de comprendre ce qui se passe lors de divers transferts de fichiers, de façon transparente (ce qui est bonne introduction à Mime). Cette commande permet de transcoder un fichier de type dit *binaire* en type *texte*.

Après avoir vu la syntaxe de cette commande dans le manuel, essayer d'encoder un fichier et voir les tailles respectives avant et après encodage.

Enfin, il est conseillé de lire au moins partiellement le manuel de `mimencode` aussi, s'il existe. On en reparlera plus tard.

4 TP3 : Internet : Correspondance entre noms et adresses

Voici la structure que l'on récupère lorsqu'on cherche à identifier un hôte :

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* host address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses from name server */
};
#define h_addr  h_addr_list[0]  /* address, for backward compatiblity */
```

Elle est décrite dans `/usr/include/netdb.h`. Un ensemble d'appels divers existe pour récupérer une structure de ce type concernant un hôte. Par exemple `gethostbyname(char *nom)` où `nom` est le nom du hôte.

Voici le début du manuel pour cet appel :

```
NAME
     gethostent,   gethostbyaddr,   gethostbyname,    sethostent,
     endhostent - get network host entry

SYNOPSIS
     #include <sys/types.h>
     #include <sys/socket.h>
     #include <netdb.h>
```

```
struct hostent *gethostent();
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const char *addr, int len, int type);
```

Exercice 5

1. Ecrire un programme qui permet d'obtenir le numéro *IP* (numéro Internet) d'une machine en fonction de son nom. Le numéro sera affiché avec la notation pointée connue.
2. Ajouter dans le programme l'affichage du nom officiel (champ `h_name` dans la structure `hostent`), ainsi que la liste des alias de l'hôte. Cette question des alias sera à discuter et à rediscuter à la suite des TP .
3. Passer le nom en paramètre dans la ligne de commande, s'il est saisi dans le programme. Ne pas oublier dans ce cas une commande de fin permettant de s'arrêter sans bavure (fin de fichier par exemple, ou `bye, quit, ...`).
4. Utiliser l'appel `gethostbyname()` et la structure `hostent` dans votre programme.
5. Essayer le programme avec des noms locaux et des noms correspondants à des sites distants, que vous connaissez dans le monde *Internet*.
6. Essayer avec des noms comme
   — `ns1.info-ufr.univ-montp2.fr`, `ns.eu.net`, d'autres,
   — `www.lirmm.fr` et `janela.lirmm.fr` ; que constate-t-on ?
   — `www.imag.fr` ; quel est son nom officiel ? `www.univ-rennes1.fr`, `www.mit.edu`, d'autres encore
7. Comparer les réponses que vous obtenez pour les noms de votre réseau local avec celles obtenues par la commande `getent`. Pour l'utilisation de cette commande, voir le manuel et l'expliquer.
8. Regarder dans le manuel l'appel `inet_addr` et les autres appels qui sont cités, comme `inet_ntoa` par exemple. Quel appel aurait pu vous rendre service pour le programme ci-dessus ? Finalement, est-ce que votre programme devient plus compliqué ou plus simple ?
9. Terminer en modifiant le programme pour afficher l'ensemble des informations contenues dans la structure `hostent`.
10. Expliquer chacune d'elles en l'appliquant aux divers hôtes cités ci-dessus.

Exercice 6

La commande `nslookup` permet de visualiser de façon interactive des informations et des caractéristiques de machines dans l' *Internet*.

1. Lancer cette commande sans paramètres.
2. Ajouter ensuite le nom complet d'un hôte quelconque et constater le résultat.
3. Refaire les questions relatives aux mêmes hôtes que ci-dessus et d'autres.
4. Comparer avec les résultats qu'on peut obtenir avec les commandes `dig` et `host`.

5 Quelques sites utiles

Les sites donnés ci-après sont relatifs à l'enregistrement de domaines et à la demande d'affectation de numéros IP. Vous pouvez les visiter pour prendre connaissance des formalités nécessaires.

Il n'est pas conseillé d'y passer trop de temps en TP pour autant.

— RIPE (`http://www.ripe.net/`).
— InterNIC (`http://www.internic.net/`)
— AFNIC (`http://www.nic.fr/`)
— ARIN (`http://www.arin.net/`).

6 TP4 : Allocation de numéros de port

Exercice 7

On peut obtenir les caractéristiques de la boîte pas l'appel système ci-dessous. Contrairement à ce qui est dit dans cet extrait du manuel, `name` n'est pas le « nom » mais contient bien l'adresse telle que définie dans le cours.

NOM getsockname - Obtenir le nom d'une socket.

SYNOPSIS
```
#include <sys/socket.h>
int  getsockname(int   s  , struct sockaddr * name , int * namelen )
```

DESCRIPTION getsockname renvoie le nom name de la socket indiquée. Le paramètre namelen doit être initialisé pour indiquer la taille de la zone mémoire pointée par name. En retour, il contiendra la taille effective (en octets) du nom renvoyé.

VALEUR RENVOYÉE getsockname renvoie 0 s'il réussit, ou -1 s'il échoue, auquel cas errno contient le code d'erreur.

Exercice 8

Ecrire un programme qui affiche le numéro de boîte réseau (numéro de port) alloué par le système, pour une boîte dite *privée*. Ce numéro est alloué lorsque le programme met à 0 (zéro) le numéro demandé. On rappelle que le champ contenant le numéro de port est `sin_port`, et qu'il faut utiliser `ntohs()`, car ces numéros sont transmis sous la forme réseau (cf Cours)

7 Un client/serveur UDP

Exercice 9

Un serveur jouant au « perroquet » tourne sur l'hôte `io.info-ufr.univ-montp2.fr`. En cas de conflit son code exécutable se trouve aussi dans le répertoire commun habituel, `/commun/info/L3/Reseau` et on peut le lancer sur tout hôte. Il ne fait qu'attendre des messages dans la boîte réseau numéro 31469, sous le protocole UDP, et renvoie le texte reçu à l'expéditeur.

1. Ecrire un client qui expédie un message et attend la réponse (l'écho en quelque sorte).
2. Si nécessaire, modifier le client pour qu'il ne se se contente pas d'expédier un seul message : il doit envoyer en boucle chaque ligne entrée par l'utilisateur, jusqu'à une demande d'arrêt.

Remarque- : les classes `Sock` et `Sockdist` sont à votre disposition dans le répertoire commun, au format source et objet. **Mais** vous devez faire le nécessaire pour ne **pas** inclure les `.cc` dans vos programmes. Il est donc demandé de construire un fichier `makefile` qui permettra de fabriquer les exécutables directement avec les `.o` des classes.

8 Communications UDP entre deux applications

Exercice 10

Soient `approlix` et `apprifix` deux applications (deux processus) sur deux hôtes distincts quelconques. Ces hôtes ne sont pas figés dans les applications (nom passé en paramètre), car on prévoit de lancer ces applications en plusieurs exemplaires sur plusieurs hôtes.

Première étape

On commencera par la mise en place d'une communication échangeant un seul message texte (chaque application envoie un message à l'autre).

On peut supposer le schéma algorithmique de base suivant pour `approlix` :

```
entier descLocal= allouerBRlocale (nomHôte, numLocal, proto) ;
initialiser BRdistante (nomDistant, numDistant, proto);
initialiser longueurBRdist;
```

```
initialiser tamponExp, longueurExp ;
initialiser tamponRec, longueurRec;
expédier (descLocal, tamponExp, longueurExp, 0, BRdistante, longueurBRdist);
recevoir (descLocal, tamponRec, longueurRec, 0, NULL, NULL);
```

On peut supposer un schéma algorithmique semblable pour `apprifix`, en inversant l'ordre d'expédition et réception.

1. Prépaper les deux programmes approlix et apprifix.
2. Que se passe-t-il si on lance une seule des deux applications ? Jusqu'où se déroule-t-elle sans encombre ?
3. Que se passe-t-il si une application fait un envoi sans que l'autre soit en réception ? Pour illuster la réponse à cette question, il faut compléter et modifier le schéma algorithmique précédent. En effet, il faut s'assurer que l'application censée recevoir soit lancée, tout en étant sûr qu'elle n'est pas en réception. Proposer une solution répondant à ce besoin. Intégrer la à votre programme

Deuxième étape

On va mettre en évidence les pertes de paquets alors que les deux applications approlix et apprifix sont lancées, c'est-à-dire que si on perd des paquets, ce n'est pas parce que le destinataire ou la boîte de destination n'existe pas.

1. Mettez en œuvre enles différents cas de figures..
2. Modifier vos programmes afin de mettre en évidence une telle perte ?
3. Créer un exemple d'interblocage des deux applications (au besoin en modifiant les programmes) ?
4. Comment débloquer par des moyens autres que l'arrêt pur et simple ?

Troisième étape

On veut qu'une seule application ait une adresse de boîte réseau connue (le numéro de service est connu) par l'autre. Par exemple `approlix` est lancée avec un numéro connu par `apprifix`, mais cette dernière se fait attribuer un numéro quelconque par le système d'exploitation local. Que proposez-vous pour la mise en place d'un dialogue sans heurts ?

9 TP5 et TP6 : Client/Serveur en TCP

Exercice 11

1. Créer deux programmes permettant d'engendrer des processus communiquant par le protocole TCP, selon le modèle *Client - Serveur*. La base de travail sera les classes `C++` distribuées précedemment et à votre disposition.

   Le Client `clientTCP` expédie un message contenant le numéro d'utilisateur ayant lancé ce processus client (voir l'appel système `getuid()`).

   Le serveur `serverTCP` lui répondra *Tiens, bonjour* suivi du nom d'utilisateur correspondant au numéro transmis (voir l'appel système vu en cours de *systèmes d'exploitation*, `getpwuid()`).

   Le serveur traite les requêtes une à une, c'est-à-dire qu'on répond complètement à un client, on ferme la connexion puis on passe au client suivant.
2. Nous avons vu que le protocole TCP était orienté *flot de caractères*. Il n'y a pas de limite de message qui soit assurée par le protocole. Du coup, il n'y a pas de correspondance entre le nombre d'envois (écritures) et le nombre de réceptions (lectures) pour un *message*.

   Est-ce que les processus que vous avez prévus tiennent compte de cette caractéristique ?Sinon, modifier vos programmes.

10 Appels système pratiques

Exercice 12

— L'appel `getsockname()`, déjà vu dans un TD précédent, permet de récupérer l'adresse de la socket (boîte réseau) dont le descripteur est `s`.

```
int  getsockname(int   s, struct sockaddr * name, int * namelen )
```

— L'appel `getpeername()` permet de récupérer l'adresse de la socket connectée à la socket `s`.

```
int  getpeername(int  s, struct sockaddr *name, int *namelen);
```

1. Un client en mode connecté peut laisser l'appel `connect()` compléter la demande d'allocation de sa boîte réseau. Écrire un programme permettant d'afficher le numéro de port réellement obtenu après avoir fait la demande `connect()`. On pourra afficher le numéro avant cette demande, pour constater que l'appel système a fait son travail.
2. Afficher sur un serveur en mode connecté les caractéristiques du client demandant une connexion.

11 Un serveur concurrent et un problème de performances

Exercice 13

On veut mettre en place un serveur de fichiers, à la manière de `ftp`, afin de faire des mesures et comparer ses performances aux outils classiques de transfert de fichiers (`ftp` ou ce même protocole sous un navigateur par exemple).

1. Ecrire un serveur et un client pour cette application. Le client ne peut que demander un fichier et le serveur le lui expédie lorsqu'il existe, ou lui annonce son inexistence : on ne prévoit pas d'autres services que le transfert de fichiers. Bien sûr plusieurs clients sont possibles et peuvent travailler simultanément. Chaque client peut demander plusieurs fichiers.
2. Est-ce que le client peut savoir s'il a reçu un fichier intégralement (est-ce prévu dans votre protocole) ?Sinon, modifier votre programme.
3. Si on veut limiter le nombre de clients simultanés, quelle solution peut-on proposer ? Mettez les en place

12 TP7 et TP8 : What a chat ?

Exercice 14

On veut mettre en place un moyen de discussion entre plusieurs utilisateurs, à la manière du logiciel IRC. Les participants à la discussion seront les *discuteurs*, tous clients d'un serveur dit *parlotteur*. Le rôle du serveur est de diffuser à tous les utilisateurs tout texte expédié par l'un d'eux.

Vous pourrez améliorer les fonctionnalités du *parlotteur*, au fur et à mesure de l'avancement du TP.

Chaque *discuteur* commence par s'identifier auprès du serveur. Son identité va servir de préfixe à tout texte qu'il désire expédier. Le préfixe est apposé par le *parlotteur*. Un *discuteur* peut se joindre à la discussion à tout moment, dès que le serveur est lancé. Il peut aussi quitter la discussion à sa guise. Lorsqu'il la quitte, il annonce sa volonté de la quitter au serveur. Bien évidemment, le premier *discuteur* devra attendre l'arrivée d'au moins un second. De même, dès qu'un *discuteur* se retrouve seul, il doit en être averti.

En principe, on veut un système de communication bidirectionnel, car nous avons vu que tout processus pouvait expédier et recevoir des messages dans une même boîte réseau. Donc, il n'est pas nécessaire de mettre en place deux communications différentes, une dans chaque sens (comme les tubes lors du cours *systèmes d'exploitation*).

Malheureusement, un problème délicat à résoudre est lié indirectement à l'interface utilisateur. En effet, dans le cas d'une interface fenêtre simple, le partage d'une fenêtre en plusieurs parties ou encore l'association d'une fenêtre spécifique à un processus n'est pas simple.

Pour éviter cette difficulté, on commencera par une simplification : chaque discuteur aura en fait deux processus à lancer, **chacun dans une fenêtre séparée** :

— un expéditeur, qui prend ce que le discuteur tape au clavier et l'expédie au serveur, bien évidemment dans le protocole d'application déterminé,

— un récepteur, qui affiche le texte reçu du serveur.

Noter que si l'utilisateur ne lance qu'une seule des deux fenêtres, le système fonctionne sans aucun problème : si l'utilisateur n'a lancé qu'une fenêtre de réception, il ne pourra pas expédier et réciproquement.

Lorsqu'on aura cette première version en fonctionnement, on pourra passer à la version bidirectionnelle.

12.1 Première étape

1. Choisir un protocole de communication au niveau de la couche transport.

2. Expliquer pourquoi il faut utiliser des entrées-sorties non bloquantes.
3. Proposer une solution cohérente et non brutale (pas de ^C par exemple) pour arrêter le dialogue.
4. Écrire les algorithmes puis les programmes correspondants.

12.2 deuxième étape

On veut passer à un fonctionnement bidirectionnel. Dans ce cas, chaque utilisateur lance un seul processus, s'attachant à la fois une fenêtre d'émission et une autre fenêtre de réception. On trouvera dans le paragraphe suivant des indications sur la gestion des fenêtres. Mais on peut opter pour toute forme d'interface qui vous convient.

12.2.1 gestion de fenêtres

Pour qu'un processus puisse obtenir l'accès à des fenêtres spécifiques, voici une solution :
— ouvrir une fenêtre de type `xterm`,
— lancer dans la fenêtre la commande `tty` qui donne l'identité associée (`/dev/ttypxx` ou `/dev/pts/xx`, `xx` étant un numéro affecté à la fenêtre),
— on peut alors ouvrir cette fenêtre par un `open()` classique ; par exemple :
`int descFen=open(/dev/ttypxx, O_RDONLY);`
— ou mieux : envoyer le nom de la fenêtre en paramètre au programme.

Mais il est clair que dans tous les cas, ce n'est pas un fonctionnement très agréable, car il oblige à trouver le nom de la fenêtre afin de le passer comme argument au processus.

Peut-on lancer une fenêtre à partir d'une application et faire des écritures dans cette fenêtre ? Avec des interfaces graphiques, oui. En générant un *émulateur de terminal*, comme `xterm` ou équivalent, mais alors il faut utiliser des options permettant d'exécuter dans cet émulateur des commandes...

12.3 troisième étape

Elle concerne plus la configuration du clavier et n'a pas d'incidence sur l'aspect *réseau* de l'application.

On veut expédier tout caractère rentré au clavier, sans attendre la validation de toute entrée par *retour chariot* (touche *entrée*).

1. Quels sont les avantages et inconvénients de cette méthode ?
2. Est-ce que le raisonnement fait ci-dessus concernant le blocage des entrées-sorties doit être modifié ?

13 TP 9 : Y a-t-il quelqu'un ? Ping

Exercice 15

La commande `ping` permet de savoir si un hôte quelconque de l'Internet est *vivant*, c'est-à-dire actif et connecté au réseau. Sa syntaxe est très simple :

`ping nom_hôte`

Il peut s'agir de n'importe quel hôte, local ou distant.

Remarque : Cette commande réagit de façon différente selon le système d'exploitation et présente plusieurs pièges, qui méritent d'être notés :
— Le premier piège consiste en général à trouver où est cette commande, car elle se trouve à des endroits différents, selon les systèmes. Si on ne l'a pas dans l'un des répertoires de recherche par défaut, on peut
  — chercher dans `/sbin`, `/usr/sbin`,
  — utiliser `which`, sinon `whereis` pour la localiser.
—

La deuxième difficulté est que la réponse peut avoir une forme différente selon les systèmes. Autrefois (?) on pouvait effectuer des essais successivement sur des machines comme `tutu, pluton` pour constater ces différences. On pourra aussi regarder le manuel sur chacun de ces hôtes.

Le protocole ICMP est à la base du fonctionnement de `ping`. Il n'y a pas de commande `pong`.

14 Routage sur Internet

Exercice 16

La commande `netstat` permet d'avoir quelques éléments concernant le routage sur une machine donnée. Attention elle n'est pas toujours localisée dans un répertoire classique et bien connu dans le monde *Unix*. On peut la chercher avec `whereis`. Exécuter cette commande en particulier avec les options `-r`, `-nr` puis `-nrs`.

Peut-on trouver ainsi le nom de la machine qui joue le rôle de routeur au niveau du réseau local ? Interpréter dans chacune des lignes affichées les colonnes *Destination, Gateway et Interface*, sans chercher à interpréter les autres. À quoi correspond le routage à destination de `127.0.0.1` ?

15 Routage encore

Exercice 17

Il existe un moyen permettant d'avoir une idée du chemin emprunté pour atteindre un hôte dans le monde Internet. C'est la commande `traceroute`. La syntaxe est aussi simple :

```
traceroute nom_hôte
```

Comme pour la précédente, il faut la localiser. Essayer de la lancer en donnant des destinations diverses, proches puis aussi lointaines que possible.

Le résultat est un chemin. Mais ce chemin peut ne pas exister. Pourquoi ? Voir le TD correspondant pour y répondre.

16 Héritage

Exercice 18

Mettre en évidence les problèmes d'héritage vus dans le TD concernant le partage des boîtes réseaux. On pourra créer un serveur qui clône et montrer que les deux processus peuvent partager les boîtes publiques ou privées ; montrer aussi qu'un message consommé par l'un n'est plus accessible à l'autre.

En particulier en utilisant *tcp* en mode connecté, expliquer pourquoi le partage de la boîte privée peut perturber lourdement le fonctionnement (penser aux messages longs).