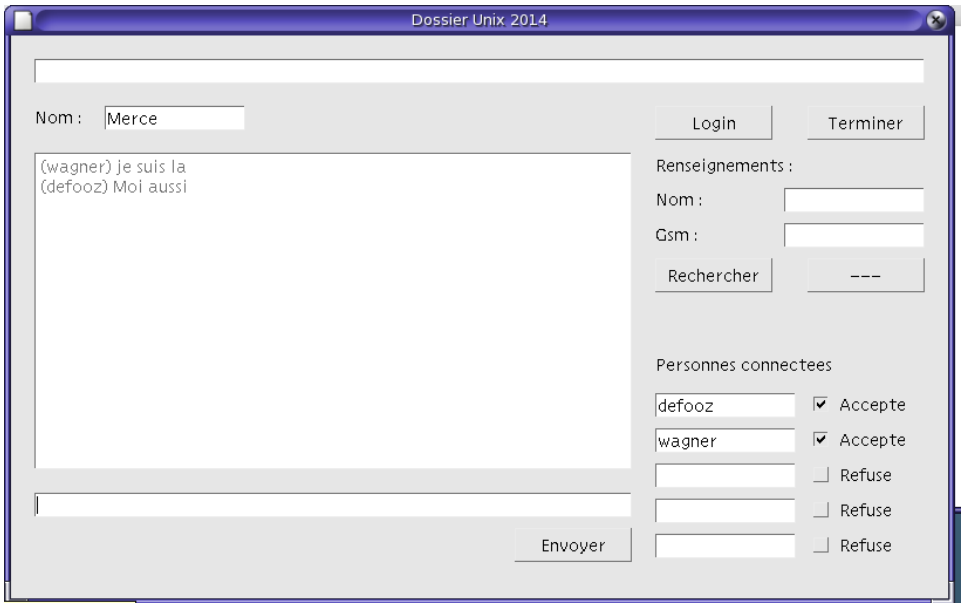
Dossier Finale Unix 2014 :

Il s'agit de communiquer entre plusieurs utilisateurs.



Une fois connectée, la personne reçoit la liste des autres personnes connectées et peut accepter ou non de lui transmettre des messages. Il ne sait pas en interdire la réception.
Il peut également demander ses renseignements sur une personne (connectée ou non) qui appartient au réseau. (pour simplifier les tests, on se contente du numéro de Gsm).
Il peut également modifier ces renseignements (donc son numéro de Gsm) mais uniquement ceux dont il est propriétaire (normal).
Une fois qu'il a terminé de « parler », il termine la communication.
ATTENTION, la fenêtre n'est pas fermée, un autre utilisateur peut alors se connecter.
Si l'utilisateur reste inactif 2 minutes, la communication est automatiquement terminée.

Pour fermer la fenêtre, il appuie sur la petite croix, et ensuite par le <ctrl-C> (**SIGINT**), il ferme la connexion.

Ce travail a pour objectif de maitriser la communication entre processus et d'illustrer les points de théorie vue au cours. Il n'est pas de programmer QT. **La solution demandée est donc purement pédagogique.**

Pendant tout le temps ou la session est active, des publicités seront affichées.

Le travail est réalisé par équipe de 2 étudiants.
Il sera défendu en janvier le jour de l'examen par un des professeurs responsables.

Réalisation :

Les sources sont, comme d'habitude, disponible dans le répertoire **/export/home/public/MerceD/DossierFinal2014**
Il s'agit

- **Client.h** et **Client.cpp** (c'est-à-dire la fenêtre)
- **moc_Client.cpp** (pour établir les connexions gérées par Qt)
- **Main.cpp** qui lance le Client
- **Fichier.ini** qui contient la définition de ce qui sera utile pour la suite le l'application.
- **Serveur.cpp** qui crée les ressources et qui gère le trafic.
- **CreeFichierUtilisateur.cpp** qui crée une dizaine d'utilisateurs
- **Compile.sh** qui permet de créer les exécutables.

Comme déjà signalé, **ne modifiez pas** les fichiers Client.h et moc_Client.cpp.

La définition des variables sera donnée à la fin du document en annexe.

Dans l'état actuel, la fenêtre lit le message et l'affiche dans la zone correspondante.

L'étape 0 consiste donc de transmettre le message au Serveur qui le retransmet immédiatement.
Pas de problème, vu que ce n'est rien d'autre que ce que vous avez réalisé dans l'exercice précédent.
Le Client émet une requête **ENVOYER** et attend la réponse pour l'afficher.

Première étape :

Le message émis doit être envoyé aux autres utilisateurs.
Le serveur doit donc connaitre tous les utilisateurs connectés.
Lorsqu'un client est lancé, il doit s'identifier.
Pour cela, il émet un message **NEWWINDOW** avec son pid. (Il devra donc émettre un message **ENDWINDOW** pour terminer la connexion)
Le serveur stocke le pid dans une table.

```
typedef struct
   { pid_t  Pid ;
      char  NomUtisateur[20] ;
      int Autre[5] ;
   } TABWINDOW ;
```

Ainsi, si 3 fenêtres sont lancées, cette table sera

```
7946  --          0     0     0     0     0
7947  --          0     0     0     0     0
7948  --          0     0     0     0     0
0     --          0     0     0     0     0
0     --          0     0     0     0     0
```

Lorsqu'un utilisateur s'identifie et est connu,
(par exemple MerceD, Wagner et ensuite Defooz) la table devient

```
7946  -Merce-          0     0     0     0     0
7947  -wagner-         0     0     0     0     0
7948  -defooz-         0     0     0     0     0
0     --               0     0     0     0     0
0     --               0     0     0     0     0
```

Ainsi, lors de l'émission d'un message, le serveur peut donc émettre un SIGUSR1 suivi du message aux différents utilisateurs. (c'est toujours le dossier précédent)

Maintenant, l'utilisateur sélectionne celui qui doit recevoir son message.
Il doit donc connaitre les différents utilisateurs connectés.
Le Serveur peut émettre un SIGUSR1 suivi du message avec le nom aux autres utilisateurs.

ATTENTION, un SIGUSR1 est utilisé pour signaler un utilisateur ou un message…

L'utilisateur doit donc émettre un message avec une requête **ACCEPTER** ou **REFUSER** .

Ainsi, toujours dans l'exemple, MerceD communique avec Wagner et Defooz, alors que Wagner et Defooz communiquent avec MerceD.

La table devient

7946	-Merce-	7947	7948	0	0	0
7947	-wagner-	7946	0	0	0	0
7948	-defooz-	7946	0	0	0	0
0	--	0	0	0	0	0
0	--	0	0	0	0	0

Pour la mise au point de l'application, ne pas hésiter à stopper un processus (**SIGSTOP**) pour le reprendre ensuite (**SIGCONT**). Ainsi, on a largement le temps d'émettre des messages et ensuite de s'énerver, car il faut les lire tous et ne pas bloquer le processus. (voir dernier point, mais vous pouvez le réaliser de suite)

Deuxième étape :

Il s'agit d'écrire une application qui tourne en continu et ne peut s'arrêter. (Ca ne doit pas arriver, mais peut toujours arriver.)
On va donc la doubler. Il y aura donc 2 Serveur qui travaillent indépendamment l'un de l'autre.

Pas de problème, il suffit d'en lancer 2.

Mais la table ci-dessus doit être commune, donc dans une mémoire partagée avec accès limité par un sémaphore (logique).
C'est un des 2 Serveur qui traite la requête, on ne peut prévoir lequel.
C'est le premier Serveur lancé qui crée les ressources, l'autre s'y attache.
Si un Serveur est stoppé ou arrêté, l'autre continue comme si de rien n'était. **Un Client ne remarque strictement rien.** Le Serveur arrêté peut reprendre ou être relancé sans problème.

Troisième étape :

Il s'agit de consulter les renseignements d'un utilisateur.

Toujours Pas de problème.

Cependant, le Serveur ne doit pas être ralenti. Pour cela, il lancera un processus **Recherche**, lui transmet la requête et se remet en attente de la requête suivante.

C'est ce processus qui traitera complètement la requête et communiquera avec le Client.

Il faut distinguer 2 cas :

- On consulte ses propres renseignements dans le but de les modifier.
- On consulte ceux d'un autre utilisateur.

Si la recherche est demandée sur un autre, il s'agit uniquement d'une recherche(**RECHERCHE**).

S'il s'agit de soi, toujours une recherche suivie d'une requête
- **MODIFIER** : si on modifie le numéro de Gsm.
- **ANNULATION** : si on ne fait rien.

Mais pendant ce temps, il est toujours possible de recevoir des messages.

Un autre utilisateur ne sait pas ce que vous faites, il peut donc

- émettre des messages
- consulter les renseignements. Mais si l'utilisateur consulté subit une modification, il ne peut pas visualiser ses renseignements (normal, ils risquent d'être faux)

→ usage des locks de fichiers.

Le processus Recherche se termine à la fin de la transaction.

Directement, s'il s'agit d'une recherche sur un autre utilisateur, à la fin de la transaction s'il s'agit d'une « modification ».

Il doit disparaître de la table des processus…

Quatrième étape :

L'affichage des publicités.

Il s'agit de transmettre des publicités aux utilisateurs.

(Rien n'est gratuit, et il faut maintenir le site…)

Pour cela, le processus Serveur lancera un processus indépendant **AffPub** qui lit toutes les 5 secondes une ligne d'un fichier texte, l'écrit dans une mémoire partagée et signale à tous les processus de lire et d'afficher la ligne.

(il peut connaitre leur pid par la M.P.)

Lorsqu'il atteint la fin de fichier, il recommence.

Cinquième étape :

Le signal SIGALRM étant libre, il peut être alors utilisé par le client pour limiter les sessions inactives pendant 2 minutes. (la session utilisateur est terminée mais pas la fenêtre)

Dernière étape : mais peut être réalisée plus tôt

Le processus **Administration**

Ce processus (attaché à la mémoire partagée) est utile pour la mise au point de l'application.

En effet, il n'est quasiment pas possible d'émettre des messages simultanément lors des tests.

On va donc suspendre un (plusieurs) processus par un **SIGSTOP**.

Et le(s)reprendre par un **SIGCONT**.

Ce programme propose un menu élémentaire avec les différents choix.

```
Serveur (1) : 9235
Serveur (2) :

Personnes connectees :
MerceD              (8464)
wagner              (8468)
defooz              (9148)

        1 - Stopper un serveur :
        2 -         un client :

        3 - Reprendre un serveur :
        4 -           un Client :

        5 - Arreter  un serveur :

        0 - Terminer
```

Noms des variables et fonctions utilisées :

```
virtual void setPersonne(int, const char*);
virtual const char* getPersonne(int) const ;
virtual void setMessage(const char*);
virtual void setPublicite(const char*);
```

et les méthodes

```
virtual void Terminer();
virtual void Rechercher();
virtual void Modifier();
virtual void Envoyer();


QLineEdit* lineNomLogin;

QLineEdit* lineNom;
QLineEdit* lineGsm;

QPushButton* ButtonLogin;
QPushButton* ButtonRechercher;
QPushButton* ButtonTerminer;
QPushButton* ButtonModifier;
QPushButton* ButtonEnvoyer;

QLineEdit* lineMessage;
QLineEdit* linePublicite;

QTextEdit* textMessageRecu;

QLineEdit* linePersonne[5];
QCheckBox* checkBox[5];
```