le cnam
École d'ingénieurs eicnam

Idriss NEUMANN
Nabi LAVERGNE

Rapport de projet
en architecture avancée de systèmes

FIP-ING39 - 2ème année spécialité Informatique à dominance SI

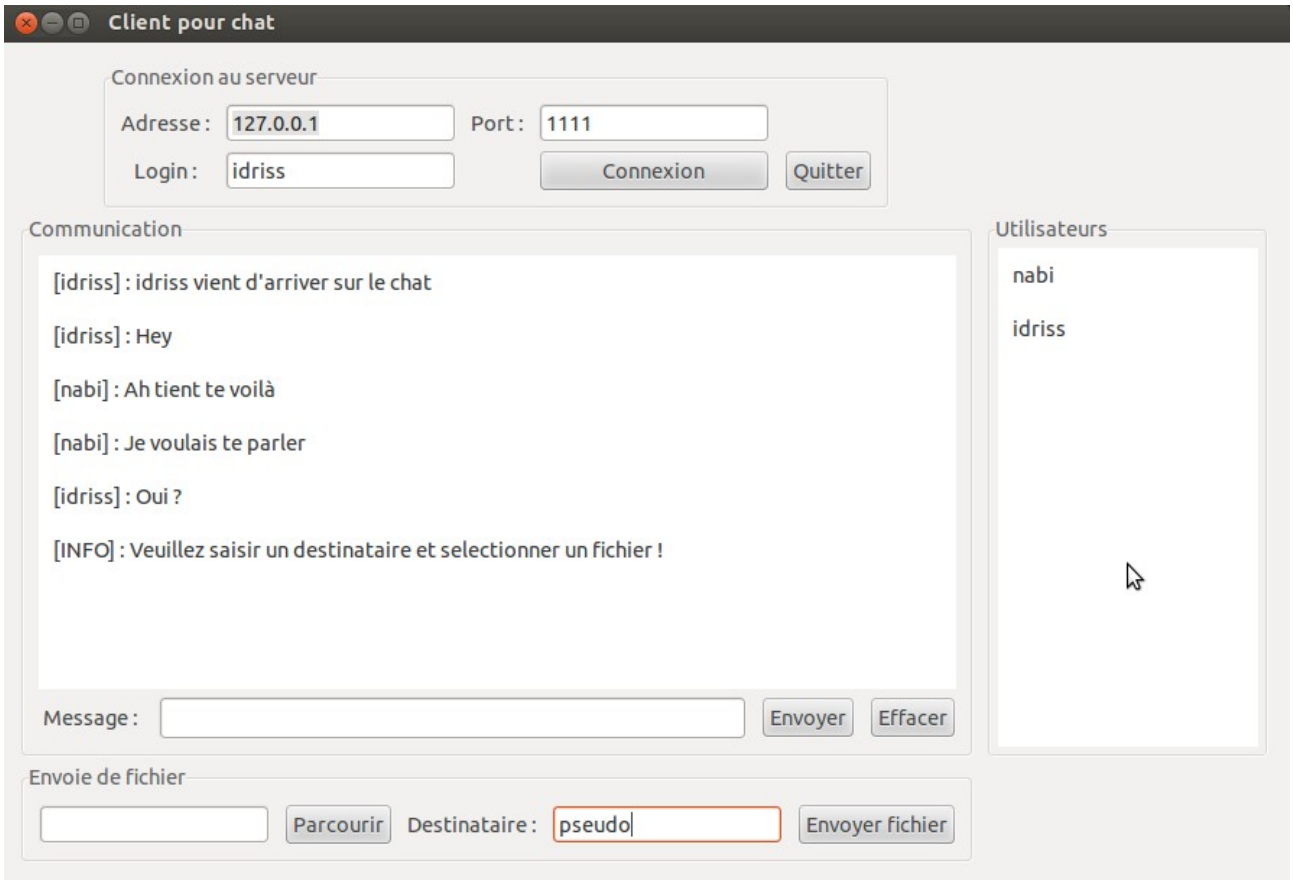
Table des matières

Introduction

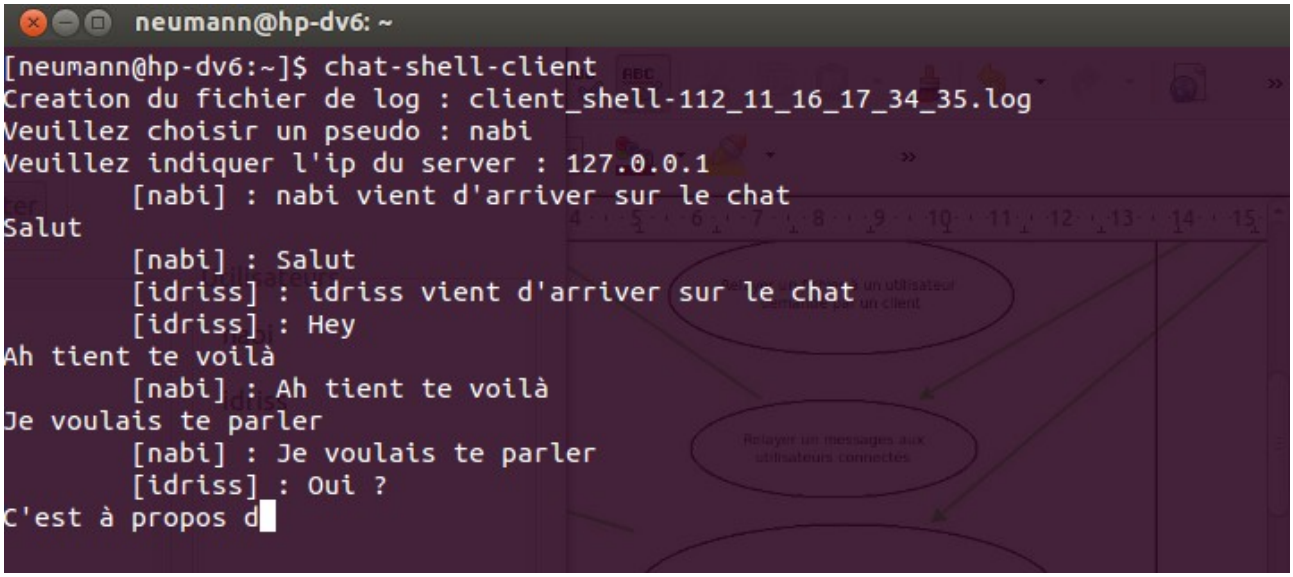
Le projet que nous avons réalisé est un ensemble de modules applicatifs permettant de mettre en place un chat avec sous architecture client-serveur. Les modules applicatifs ont tous été développés en C POSIX avec la bibliothèque graphique GTK (vous trouverez les détails des compétences utilisées dans la partie III) et sont les suivants :

- Le serveur ;
- Le client en mode graphique ;
- Le client en mode shell (ou en mode console, sans interface graphique).

Capture d'écran du client graphique :



Capture d'écran du client shell (en mode console) :



Ces modules applicatifs permettent de communiquer entre plusieurs clients et de s'échanger des fichiers. L'ensemble des fonctionnalités seront détaillés dans la partie I des spécifications fonctionnelles.

I – Spécifications fonctionnelles

Dans cette partie, nous allons détailler les spécifications fonctionnelles des différents modules applicatifs à l'aide d'un diagramme des cas d'utilisations en UML. Ces spécifications fonctionnelles permettront de répondre à la question «que doivent faire ces modules applicatifs ?».

Le diagramme des cas d'utilisations général liste l'ensemble des fonctionnalités des différents modules applicatifs :

II – Conception détaillée

Dans cette partie, nous détaillerons comment l'application répond t-elle aux spécifications fonctionnelles à l'aide d'exemples de scénarios et de diagrammes de séquence.

1) Scénario nominal pour le client

1 – l'utilisateur saisit les informations nécessaires à la connexion (IP, port et pseudo) et valide ;

2 – le client ouvre une connexion TCP au serveur qui vérifie si le pseudo est valide et n'est pas utilisé ;

3 – le serveur envoie la liste des utilisateurs connectés au client ;

4 – le serveur envoie à tout le monde l'information de connexion du client ;

5 – l'utilisateur dialogue avec les autres clients connectés et échange des fichiers ;

6 – Fin.

2) Scénario d'alternative «le pseudo est déjà utilisé»

3 – le serveur envoie l'information permettant au client de savoir que le pseudo est déjà utilisé ;

4 – le client affiche que le pseudo est déjà utilisé ;

5 – l'utilisateur saisit un nouveau pseudo puis valide ;

6 – retour au 2) du scénario nominal.

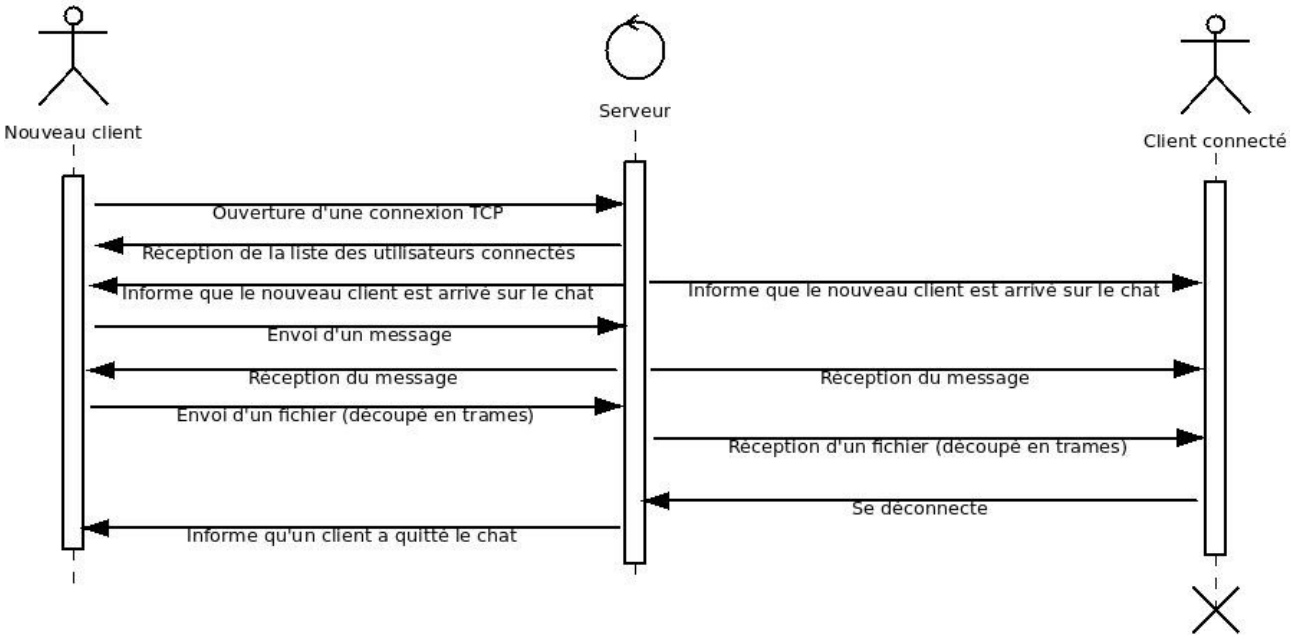
3) Scénario d'exception «le serveur n'est pas joignable»

2 – le client n'arrive pas à joindre le serveur (IP invalide, port invalide, …) ;

3 – le client affiche «Connect error !» sur stderr puis quitte en erreur ;

4 – Fin.

4) Exemple de diagramme de séquence



III – Choix d'implémentation

1) Matrice des compétences par modules applicatifs

Cette matrice résume les concepts ayant été utilisés dans l'implémentation des différents modules applicatifs :

Compétences / Modules applicatifs	M1 - Serveur	M2 - Client version graphique	M3 - Client version shell
C1 – Programmation en C POSIX	**X**	**X**	**X**
C2 – sockets TCP	**X**	**X**	**X**
C3 – fork			**X**
C4 – pthread	**X**	**X**	
C5 – IPC shared memory			**X**
C6 – interface GTK		**X**	
C7 – utilisation de select	**X**	**X**	**X**
C8 – manipulation de fichiers binaires et textes	**X**	**X**	**X**
C9 – manipulation de listes chaînées	**X**	**X**	**X**

2) Rôles des différents fichiers sources

Les librairies utilisées

Fichiers	**Rôle**
Entête : listhead.h **Fonctions :** listfunctions.h	Structure, maccros et fonctions relatives à la manipulation de liste chaînée. Cette lib sert à la fois pour les listes de clients connectés sur les différents clients et le serveur ainsi que les listes des fichiers en cours de téléchargement sur chaque client.
Entête : loghead.h **Fonctions :** logfunctions.c	Fonctions relatives à l'écriture dans des fichiers de logs repris sur le modèle de syslog (avec un niveau de log permettant de faire des recherches et filtrage sur les types de messages recherchés).
Entête : sockethead.h **Fonctions :** socketfunctions.c	Structure, maccros et fonctions relatives aux sockets. Cette lib encapsule les primitives POSIX de la lib socket et des IPC permettant d'établir des communications TCP, UDP ou encore Unix (files de messages) en toute simplicité.
Entête : iohead.h **Fonctions :** iofunctions.c	Fonctions relatives aux entrées sorties (saisie sécurisée sur stdin et vider le buffer stdin).

Modules applicatifs

Fichiers	Rôles
server.c	Fonctions et programme principal du serveur. Celui-ci permet de relayer les messages des différents clients connectés ainsi que les fichiers envoyé par un client pour un autre client.
client.c	Fonctions et et programme principal du client graphique. Ce dernier permet d'échanger des messages, d'envoyer et recevoir des fichiers.
client_shell.c	Fonctions et programme principal du client en mode console. Ce dernier permet d'échanger des messages et de recevoir des fichiers envoyés par un autre client (graphique).

3) Extraits de codes

Dans cette partie nous fournirons quelques extraits de codes afin de détailler davantage certains choix d'implémentation.

TODO :

Fonctions TCP de la librairie socketfunctions.c

Fonctions de manipulation des listes chaînées

Thread client et ouverture de la thread sur le serveur

Thread d'écoute sur le client graphique

Fork et processus d'écoute sur le client shell

IV – Installation et utilisation

Dans cette partie, nous allons détailler la procédure d'installation et d'utilisation des différents modules applicatifs.

1) Installation standard

Prérequis :

- Linux (le logiciel fonctionne très bien sur Ubuntu 12.04 LTS 64 bits avec Gnome/Unity ;
- Le package libgtk2.0-dev sous Ubuntu ou équivalents pour les autres distributions ;
- /bin/bash comme shell par défaut ;
- gcc et make (packages build-essential pour Ubuntu).

Vérifications des prérequis :

```
[ $] echo $SHELL # vérifier que à renvoit bien /bin/bash
[ $] sudo dpkg -l | grep build-essential # vérifier que le package est bien installé
[ $] sudo dpkg -l | grep libgtk2 # vérifier que le package est bien installé
```

Installation des prérequis en cas de besoin (pour Ubuntu) :

```
[ $] apt-get update
[ $] apt-get install build-essential
[ $] apt-get install libgtk2.0-dev
```

Installation des modules applicatifs :

```
[ $] tar xfvz chat_neumann_lavergne.tar.gz
[ $] cd chat/
[ chat$] chmod +x configure.sh
[ chat$] ./configure.sh
```

Lancement :

```
[ $] chat-server
[ $] chat-client
[ $] chat-shell-client
```

2) Installation dégradée

Dans cette étape vous pouvez choisir quoi compiler sans installer l'exécutable dans votre PATH. Cela peut s'avérer utile si vous n'avez pas Gtk d'installé et que vous souhaitez utiliser la version shell ou si vous n'avez pas /bin/bash comme shell par défaut.

Prérequis :

- Linux ;
- gcc et make (package build-essential sous Ubuntu).

Vérifications des prérequis :

```
[ $] sudo dpkg -l | grep build-essential # vérifier que le package est bien installé
```

Installation des prérequis en cas de besoin (pour Ubuntu) :

```
[ $] apt-get update
[ $] apt-get install build-essential
```

Compilation des modules applicatifs :

```
[ $] tar xfvz chat.tar.gz
[ $]cd chat/
[ chat$] make server
[ chat$] make client
[ chat$] make client_shell
```

Ou si vous souhaitez tout compiler d'un seul coup :

```
[ chat$] make
```

Lancement :

```
[ chat$] ./server
[ chat$] ./client
[ chat$] ./client_shell
```

3) Utilisation

Pour les différents modes d'installation choisis, il faut :

- Lancer en premier le server (comme indiqué dans les parties précédentes) ;
- Attendre que le server renvoi "Server is ready" ;
- Lancer un client et se connecter avec un pseudo et l'ip 127.0.0.1 pour une installation locale ;
- Si le client renvoit "Connect error", vérifier que le server est bien lancé.

Les différentes commandes du chat :

- **`/QUIT`** : quitter le chat (l'info de déconnexion est envoyée au server) ... cela revient à fermer l'IHM sur la version graphique ;
- **`/STOP`** : stop le server à distance.

Des fichiers de logs sont disponibles pour chaque utilisation. Les fichiers concernés se trouvent dans le répertoire dans lequel vous lancez les commandes, ils ont pour noms :

- **`server-<date>.log`** : fichier de log server ;
- **`client-<date>.log`** : fichier de log client ;
- **`client_shell-<date>.log`** : fichier de log pour le client en mode shell.

Des levels vous permettent de cibler vos recherche dans ces fichiers de logs à l'aide de la commande grep par exemple.

Les différents levels sont les suivants :

- **`LOG_DEBUG`** : pour mettre des traces pour debugger
- **`LOG_ERR`** : pour les erreurs
- **`LOG_INFO`** : garde trace des échanges
- **`LOG_WARNING`** : avertissements

4) Avertissements

- Ne pas utiliser les exécutables déjà présents dans l'archive .tar.gz (fonctionnent pour une Ubuntu 12.04 LTS en 64 bits). Une fois le script configure.sh lancé, l'archive .tar.gz est reconstruite avec les nouveaux exécutables compilés chez vous ;
- Pour vérifier si le port 1111 est occupé :
  ```
  [ $] sudo netstat -anpe | grep 1111 | grep LISTEN | cut -d "/" -f2
  ```

Conclusion

Le projet est bien parvenu à son terme : toutes les spécifications fonctionnelles présentées précédemment ont été implémentées et testées sur Ubuntu 12.04 LTS.

Des optimisations pourraient être réalisées notamment au niveau de la gestion des threads sur le serveur. Actuellement le serveur gère une thread par client. On aurait pu choisir de gérer les threads par clients et par réceptions et traitements de messages en faisant bien attention de calculer un nombre de threads simultanés en fonction du nombre de CPU de la machine cible.