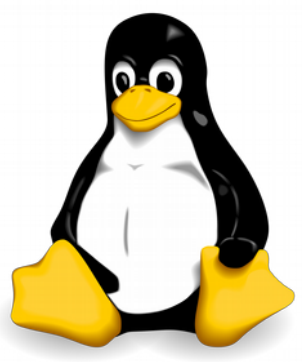
Aubry Alexandre & Bouscarrat Léo G1

07/06/2016



Rapport

-

Projet Programmation des Systèmes d'Exploitation

Échanges de messages et de fichiers cryptés

(Guide de l’utilisateur page 4+)

Première Partie : Sujet

L’objectif du projet était de créer un programme qui permettait d’appliquer les connaissances du cours. Nous avons choisi de créer une application avec un serveur et plusieurs clients. Le sujet choisi pour ces deux programmes client et serveur est un échange de messages et de fichiers cryptés entre les clients via le serveur.

Les utilisateurs se connectent au serveur et choisissent un nom d’utilisateur pour pouvoir communiquer entre eux.

Le serveur et un utilisateur sont reliés par un socket. Il y a donc autant de canaux que d’utilisateurs.

Par ailleurs, nous avons choisi que la partie cryptage/décryptage soit gérée par les utilisateurs mais que le serveur soit le seul à pouvoir générer des clés pour les opérations de cryptographie (ce qui pourrait être le cas dans une utilisation réelle si on suppose que le programme du serveur est plus performant que la normale en terme d’aléatoire et que le cryptage asymétrique n’existait pas …). Ces clés sont symétriques et seront transmises en clair durant toutes les opérations. L’algorithme de cryptage qui sera implémenté est une version simplifiée de l’AES-256, disponible via libssl-dev.

Le serveur doit maintenir une liste d’utilisateur à jour pendant toute sa durée de fonctionnement pour assurer de la fiabilité des transferts.

Un log est rempli par le serveur pour garder des traces des métadonnées. On garde des informations de connexion, d’appel de fonctions, de tentative de communication entre plusieurs utilisateurs et bien sûr les déconnexions.

Seconde partie : Organisation des programmes

Le programme est organisé entre le serveur et le client. Pour représenter les utilisateurs, nous avons créés une structure contenant, le pseudonyme, l'état (connecté ou déconnecté), un flag nous servant pour les communications internes entre worker et un message allant de paire avec le flag.

Côté serveur :

- Dans un premier temps : il écoute si il y a des connexions, dès qu'il y en a une il crée un worker pour le nouvel arrivant.
- Dans le worker : dans un premier temps, le worker gère la connexion, il crée une nouvelle case sur la liste des utilisateurs et le passe en état connecté. Ensuite il rentre dans une boucle où il écoute sans cesse le pipe avec lequel il est connecté. Il reconnaît ce qui est demandé par le client et agit en fonction.
  - Pour afficher la liste des utilisateurs, il suffit de faire une boucle sur la liste des utilisateurs et d'afficher ceux ayant l'état connecté.

- Pour envoyer un message, on va vérifier si l'id est valide. Ensuite on va vérifier si l'autre utilisateur est ouvert à recevoir ce message, pour ceci on active son flag et on change son message.
- On fonctionne de la même façon pour le fichier, néanmoins, il est nécessaire de faire de nombreux aller retour entre les deux clients, ce qui alourdit considérablement le code.

- Dans le worker : il regarde aussi si son flag n'est pas activé. Ainsi si il l'est il va lire le message et l'envoyer vers son client. Selon les cas, il va y avoir plusieurs allers-retours afin de faire passer des messages.

Côté client :

- On s'occupe d'abord de la connexion, c'est-à-dire du choix de son pseudonyme.
- Ensuite, on affiche un menu puis on utilise la fonction select afin d'écouter deux choses en parallèle :
  - La pipe reliant le serveur et le client, afin de voir si le serveur ne veut pas nous faire passer des informations.
  - Et stdin pour voir si l'utilisateur ne demande pas d'activer un des menus que nous avons affiché.

Troisième partie : Multiexécution et gestion de la concurrence

Les programmes clients ne sont pas sujets à du multithreading ou de la concurrence en interne. Une gestion événementielle est toutefois implémentée pour pouvoir gérer la réception d'un message pendant que d'autres commandes sont en attente.

Le serveur est une application du multithreading. Comme dit en partie 1, il y a autant de canaux que d'utilisateurs. Chaque canal est géré après sa création par le thread main par un thread fils. Le log est maintenu par chaque thread fils.

Un thread supplémentaire est créé par le serveur : threadMdp. Ce thread génère aléatoirement des mots de passe et recommence cette action toutes les 1 ms. De telle sorte qu'il est hautement improbable que deux utilisateurs demandent un mot de passe en même temps et qu'il s'agisse du même. La concurrence est aussi implémenté dans ce thread. La chaîne de caractères qui contient le mot de passe est protégée en lecture pendant que la thread régénère le mot de passe.

Quatrième partie : Bilan du projet

Tout d'abord ce projet nous a permis d'utiliser un nouvel outil pour le travail de groupe qui est GitHub. En effet, la programmation à plusieurs est en général très peu efficace et on perd beaucoup de temps à comprendre et retrouver les changements qu'a effectué l'autre binôme. Néanmoins quand une difficulté majeure se présentait nous pouvions nous réunir et trouvé une solution à deux.

Cela fut le cas lorsqu'il a fallu mettre en place la gestion événementielle dans le client pour gérer les entrées-sorties des fonctions 2 et 3.

Malgré le fait que la fonction finale soit quasiment complète, nous aurions voulu améliorer l'utilisation de sous-fonctions dans les deux programmes ainsi que mieux contrôler les erreurs et améliorer l'interface pour rendre le programme plus convivial. Nous estimons avoir beaucoup progresser depuis le début des travaux pratiques et le rendu de ce projet.

Guide d’utilisateur

1. Installation du programme et compilation

Pré-requis :

-terminal Linux
-3 Mo d’espace disque
-un fichier à crypter (utiliser l’image fournie)
-build-essential installé

Installer la librairie libssl-dev (dépôts standards d’Ubuntu ou supportés par Canonical) :

- sudo apt-get install libssl-dev

Pour compiler les programmes :

-make dans un terminal ouvert dans le dossier projet

2. Lancement du programme compilé

Pour exécuter les programmes :

-copier les fichiers « serveur » et « client » du dossier projet dans les dossiers portant leurs noms
-ouvrir tout d’abord un terminal dans le dossier /serveur et lancer la commande ./serveur 5000
-ouvrir un ou plusieurs terminaux dans le dossier /client et lancer la commande ./client localhost 5000

3. Connexion et liste des fonctions

Lorsque l’on démarre la connexion depuis un client, il faut choisir un pseudo pour s’identifier.

On a accès à 5 fonctions :

1. La première fonction permet d’afficher la liste des utilisateurs actuellement connectés ainsi que leur id. Il est fortement recommandé d’exécuter cette fonction avant les fonctions 2 et 3.

2. La seconde fonction permet au client d’envoyer un message à un autre client actif. Elle demande d’abord la permission au client cible puis permet de saisir le message. **Le client receveur doit appuyer sur entrée pour rafraîchir le buffer des messages reçus.**

3. Cette troisième fonction est la plus complexe. Elle demande l’autorisation à un autre client pour lui transférer un fichier (linux.png) et si c’est accepté, elle crypte le fichier, le transfert vers le second client. Ce dernier décrypte ensuite le fichier. **Le client receveur doit appuyer sur entrée pour rafraîchir le buffer des messages reçus.**

4. Cette fonctionnalité permet de tester le cryptage d’un fichier linux.png situé dans le dossier du client. Le client demande un mot de passe au serveur. Le fichier de sortie s’appelle crypto.dat.

5. Cette fonction ne peut être appelée que si la fonction 4 a déjà été appelée par le client dans la même session. Elle décrypte le fichier crypto.dat en decrypto.png.