BEN LEMRID Samy BLOIS Benjamin

Rapport du projet C

VoIP

Table des matières

Compilation du projet

Le projet se décompose en deux exécutables : un client et un serveur.
Un Makefile est disponible dans src/ pour compiler le projet, voici les paramètres qu'il accepte :

« client » permet de ne compiler que la partie client.

« serveur » permet de ne compiler que la partie serveur.

« clean » permet de supprimer tout les fichier objets (.o) et les fichiers temporaires (~), mais laisse les executables.

« cleanmax » agit comme clean, mais supprime également les exécutables.

Sans paramètres, le makefile compile l'intégralité du projet, client et serveur.

La structure du makefile est un peu particulière et mérite d'être quelque peu expliquée.
Nous utilisons deux flags de compilation « CLIENT » et « SERVEUR », permettant de faire de la compilation conditionnelle dans le code a l'aide de #ifdef CLIENT (resp. SERVEUR).
Cela se traduit au sein du makefile en 3 listes de fichiers a générer : la liste OBJ, qui contient les fichiers dans lesquels la compilation conditionnelle n'est pas nécessaire, SERVEUR dans lesquels le flag SERVEUR doit être utilisé a la compilation, et enfin la liste CLIENT, dans lesquels le flag CLIENT est nécessaire.
On trouvera également dans les liste CLIENT et SERVEUR les fichiers utilisés dans seulement un des exécutables.
Les fichiers de la liste objet sont compilés en objet sans plus de traitement.
Les fichiers dans la liste SERVEUR (resp CLIENT) sont compilés sous la forme <fichier>_serveur.o (resp _client.o pour CLIENT).
Enfin, pour la création de l'executable serveur (resp client), les fichiers objets générés par la liste OBJ sont utilisés, plus les _serveur.o générés par la liste SERVEUR (et on ne touche pas aux _client.o).

Pour compiler, les bibliothèques pthread, lasound et gtk+2 sont nécessaires, respectivement pour l'utilisation des threads, de ALSA et pour l'interface graphique.
Ces bibliothèques sont généralement présentes par défaut sur les distributions récentes de Linux, ou sinon peuvent être facilement récupérées a l'aide d'un gestionnaire de paquet (ou pour gtk+2 directement sur le site de gtk).
Nous utilisons également l'outil pkg-config, permettant de grandement simplifier les inclusions nécessaires pour gtk, mais encore une fois cet outil est disponible par défaut sur les distributions récentes de Linux.

Lancement du projet

Pour lancer le projet, il suffit de faire la commande ./client ou ./serveur selon l'utilisation.
Le programme ne prend pas de paramètres particuliers (ces derniers étant passés a gtk a son initialisation).
L'interface graphique est alors lancée, et tout s'y passe alors.
Des affichages peuvent avoir lieux sur la console, mais ils sont uniquement a titre informatif, ou d'erreur, et ne sont pas utiles dans une utilisation normale.

Utilisation du programme

Une fois le programme lancé comme expliqué dans la partie précédente, tout se passe sur l'interface.
Les interface client et serveur sont très semblables et fonctionnent de la même manière.
Il y a deux champs textes en haut de l'interface, une contenant l'adresse du destinataire (pour le client) ou du serveur (pour le serveur), et la deuxième le port a utiliser.
Pour l'adresse, le programme accepte aussi bien une adresse ip, un nom d'hôte, une URL, etc.
La résolution en adresse exploitable par le programme est faite automatiquement, et si elle n'est pas possible, l'utilisateur est retourné sur la page et invité a essayer une autre adresse utilisant sont adresse locale (de type 192.168.0.1). Il existe une adresse acceptant la connexion peu importe la valeur utilisée, c'est 0.0.0.0 et c'est l'adresse renseignée par défaut pour le serveur.
Il est aussi important de noter que pour le serveur, l'adresse renseignée est la seule par laquelle le serveur sera accessible.
Exemple si on lance le serveur 127.0.0.1,il ne sera accessible que depuis le loopback local, et non depuis une machine du même sous réseau en
Pour le port, bien qu'aucun filtrage actif n'ait été mis en place, seul une suite de chiffre ne résultera pas sur une erreur de résolution avec retour sur la page et invitation a ressayer. De plus, il est conseillé de prendre un numéro de port supérieur a 1000, les ports étant souvent réservés et donc inutilisables en dessous de cette limite.

Une fois l'adresse du destinataire ou du serveur et le port renseignés, il ne reste plus qu'a demander la connexion en cliquant sur le bouton « connexion ».
En cas d'erreur (adresse ou port invalides, destinataire inaccessible, etc...), un message d'erreur s'affiche et l'utilisateur est invité a réessayer de se connecter en changeant les paramètres.
En cas de succès, le client tentera de se connecter au serveur a l'adresse renseignée, tandis que le serveur se mettra en attente de demande de connexion sur l'adresse renseignée.
Si le client et le serveur ont rentrés les connes adresses, la connexion s'établit automatiquement et la voix est transmise dans les deux sens.
Attention, aucun filtrage n'étant effectué sur le son en entrée, l'utilisation d'un micro/casque est fortement recommandée pour éviter un effet Larsen très désagréable.

Quand les deux personnes en communication souhaitent arrêter la connexion, elles doivent toutes deux cliquer sur déconnexion.
Cela coupe l'écoute pour le serveur et la connexion pour le client, ainsi que l'enregistrement/lecture du son et l'envoi des paquets pour les deux paires.

Difficultés et résolution

Au début, le client se connectait directement au serveur à travers l'interface. Il n'y avait pas de filtrage des connexions. Pour palier à ce problème, il a été nécessaire de mettre en place un serveur TCP afin de mieux gérer l'établissement des connexions.
Avec ce système, chaque demande de connexions soumise au serveur peut être traitée ou rejetée.

Quand un projet regroupe plusieurs programmes qui traitent chacun une tâche indépendamment, l'utilisation de threads est fortement conseillé. En effet, les processus de capture et de lecture du son doivent être traités en parallèle. Sinon, le signal audio transmis n'est pas correcte.

La gestion des bibliothèques Linux audio ALSA et réseau Socket ainsi que la maîtrise de leurs différentes API ont constitué un travail particulièrement difficile.
En effet, ces bibliothèques regroupent un panel important de fonctions.
Avant de se lancer dans la programmation, il est important de bien connaître l'utilisation chaque API.

A chaque fois que le client communique avec le serveur, il envoie des paquets de données audio au format PCM non compressé. Au cours de la communication, certains de ces paquets ne sont pas correctement envoyés. L'impact causé par ces pertes de données est minime.
L'utilisateur à l'autre bout de la connexion parvient comme même à entendre l'intégralité du message transmis par le client. Pour traité ce problème, il faut indiquer au serveur quelles ont été les paquets perdues et les lui renvoyer. La gestion de l'envoi/ réception des paquets s'effectue à travers le protocole UDP. L'ordonnancement des paquets se fait par l'intermédiaire d'un protocole
pseudo-RTP. Ce dernier permet de gérer la synchronisation des paquets.
Pour ne plus avoir de perte de paquets, il faudrait améliorer ce protocole RTP en lui intégrant une fonction qui avertit l'utilisateur à chaque fois qu'un paquet n'a pas été correctement envoyé, puis une fois que ce dernier est récupéré, le renvoyé à nouveau à son destinateur.

Choix techniques

Nous allons dans cette partie détailler les technologies et techniques utilisées dans ce projet, et pourquoi nous les utilisons.

ALSA :

C'est la technologie d'enregistrement / lecture du son proposée dans le sujet. Elle a l'inconvénient de ne pas être portable, et sa documentation mériterais d'être plus claire, mais possède l'avantage de s'utiliser plutot facilement. Nous avons considéré OpenAL comme potentielle remplaçante qui possède l'avantage d'être portable, mais ayant appris tardivement que nous pouvions choisir notre bibliotheque son, nous avons préférés ne pas changer en route. De plus, ALSA est disponible par défaut sur la plupart des distribution Linux nativement, ce qui n'est pas le cas de OpenAL (bien qu'elle puisse s'installer facilement), et donc aurait compliqué l'installation pour les utilisateur.

Socket UNIX udp:

Ici, la technologie s'imposant d'elle même. En effet c'est le moyen le plus facile si ce n'est le seul pour communiquer entre plusieurs machine au sein d'un réseau. Le protocole UDP était indiqué dans le sujet et est le plus adapté a un transfert en masse ou la fiabilité de connexion n'est pas primordiale.

Pthread, ou threading UNIX :

Nous utilisons principalement le threading pour garder la main sur l'interface graphique pendant la communication, permettant ainsi de la couper a tout moment avec le bouton déconnexion. Il y a actuellement deux threads, un servant a l'écoute / envoi des paquets de son, et l'autre a la réception / lecture des paquets. Idéalement, deux threads seraient nécessaires pour cette dernière étape : un pour la réception et l'ordonnancement des paquets dans la liste, et l'autre pour la lecture de la liste de manière circulaire sans regard sur l'arrivée des paquets.
Nous utilisons également un thread de demande de connexion, afin d'établir une connexion propre et voulue par les deux pairs avant de lancer la communication.

Gtk+2 :

Pour faire une interface graphique nous avions plusieurs choix possibles : utiliser une bibliothèque graphique basique comme la SDL, ou OpenGL, ou une bibliothèque de création d'interface comme GTK ou Qt. Nous avons préférés cette deuxième solution pour trois raisons. Premièrement, ce deuxième type implémente de nombreux wigdet (boutons, champs texte...) permettant de faire une interface plus convivial. Deuxièmement, l'utilisation de ces librairies

graphiques permettent l'harmonisation du parc logiciel disponible sous Linux, souvent considéré comme trop hétéroclite. Et enfin, troisièmement et surtout, nous n'avions jamais utilisés ce type d'interface en C, sans orientation objet, et voulions voir comment cela marchait et nous y habituer.
Il nous restait a faire un choix sur la bibliothèque a utiliser. Qt n'ayant pas de portage C, nous ne pouvions l'utiliser, il restait alors le choix entre gtk2 et gtk3. Gtk3 n'était pas très vieux, et donc pas implémenté sur certaines machines, nous avons préférés rester sur gtk+2, disponible sur plus de machines.

Executables et compilation séparés par Flag :

Nous souhaitions a l'origine mettre en place une application multiclient - serveur, ou le serveur ne sert que d'annuaire et d'intermédiaire. Deux executables étaient alors nécessaires.
De plus , il est évident que certains bouts de codes se répéteraient entre les deux parties. Plutôt que de les répéter, nous avons décidés d'utiliser le même code dans les deux programmes. Enfin, parfois, seul une toute petite partie diffère entre client et serveur. Pour rester pragmatique et éviter de dupliquer des bouts de codes, il fallait un moyen de faire la distinction, et ne voulant pas faire de tests a répétition, nous avons décidé d'utiliser des Flag de compilation.