Projet de Systèmes (L3 Info)

Un système de streaming audio

Abstract

Le but de ce projet est d'écrire un système de streaming audio similaire à RealPlayer ou Windows Media Player. Le système devra transmettre des fichiers audio à travers le réseau, et jouer le fichier audio *pendant* le tansfert des fichiers. Dans un second temps nous rajouterons des filtres qui modifient le son des fichiers audio. Enfin, vous devrez écrire une petite documentation pour expliquer comment fonctionne votre système.

Pour réaliser ce projet, vous trouverez la librairie audio (décrite pendant de 3e cours de SYR2) dans le répertoire `/share/l3info/syr2/proj/`. **Lisez attentivement le fichier README qui vous explique comment utiliser le programme `padsp` (présent dans la librairie) pour faire fonctionner la sortie audio.** Vous trouverez également un petit fichier WAV que vous pouvez utiliser pour tester vos programmes (vous avez bien sûr le droit d'utiliser d'autres fichiers WAV).

ATTENTION: padsp semble avoir un comportement capricieux cette année. Si votre programme ne fonctionne pas parce que write() échoue à cause de *"connection reset by peer"*, ce n'est pas forcément un bug de votre programme. Essayez simplement de relancer le programme quelques fois.

Le système de streaming audio se composera de deux parties. Un serveur audio possèdera un certain nombre de fichiers qu'il peut proposer. Il transfère ces fichiers à la demande à travers le réseau. Un client permet d'envoyer une requête au serveur pour jouer l'un de ses fichiers. Il reçoit le flux de données à travers le réseau, et joue la musique sur le haut-parleur *pendant* qu'il reçoit les données. La communication entre le client et le serveur se fera à l'aide d'un petit protocole que vous devrez concevoir. Ce protocole sera implémenté au-dessus d'UDP uniquement. L'utilisation de TCP est interdite dans ce projet.

Le serveur et le client pourront interposer des *filtres* qui modifient les données transférées. Vous devrez écrire au moins trois filtres différents de votre choix.

Pour vous guider dans le développement de ce système relativement complexe, nous vous proposons un certain nombre détapes qui devront être réalisées à une certaine date.

A rendre pour le Vendredi 1er mars 2019: un lecteur audio

Nous commencerons par prendre en main la librairie audio. Pour ce faire, vous devrez écrire un programme `lecteur.c` (et son Makefile) qui lit un fichier audio dont le nom est passé par la ligne de commande, et qui le joue sur la sortie audio de votre ordinateur. Ce programme n'implémente pas de fonction réseau.

Vous devrez également répondre aux question suivantes:

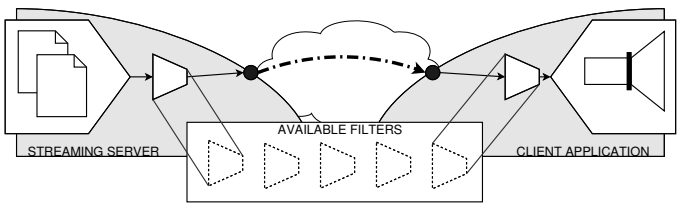

Figure 1: Organisation d'un système de streaming audio

1. Que se passe-t-il si on déclare un fausse fréquence d'échantillonage à la sortie audio? Vous pouvez essayer de passer une fréquence deux frois trop importante, ou deux fois trop faible. Expliquez le phénomène obtenu.

2. Que se passe-t-il si vous déclarez à la sortie audio que le fichier est mono, alors qu'il est en réalité en stéréo? Expliquez le phénomène obtenu.

3. Que se passe-t-il si vous déclarez à la sortie audio une mauvaise taille d'échantillons (8 bits à la place de 16 bits)? Expliquez le phénomène obtenu.

A rendre pour le vendredi 22 mars 2019: streaming audio

Nous voulons maintenant écrire un programme serveur et un programme client.

- Le serveur s'appelle `audioserver`. Il ne doit *pas* prendre de paramètres de ligne de commande. Ce programme s'attend à trouver un ou plusieurs fichiers WAV dans son répertoire local. Il attend qu'un client lui envoie une requête pour lire l'un de ses fichiers. Quand il reçoit une telle requête, il ouvre le fichier en question et commence son transfert vers le client. Quand le fichier est entièrement lu, il ferme le fichier et attend la prochaine requête client.

- Le client s'appelle `audioclient`. Il *doit* avoir l'interface suivante:
  `audioclient server_host_name file_name`
  Le client envoie une requête au serveur contenant le nom du fichier qu'il souhaite jouer. Il reçoit le flux de données du serveur et il joue le son *pendant* qu'il reçoit les données.

La communication entre votre client et votre serveur *doit* être implémentée à l'aide de datagrammes UDP. L'utilisation de TCP est interdite. Vous devrez définir vote protocole vous-mêmes. En particulier il est important que le serveur envoie les données au client à la bonne vitesse pour que le client puisse les jouer convenablement. Quelles informations doivent être transmises dans chaque datagramme? Que doit faire chaque programme quand il reçoit un tel message?

Vous pouvez faire les hypothèses suivantes:

1. Le premier paquet transmis du client vers le serveur n'est jamais perdu par le réseau. De même, le premier paquet transmis dans l'autre sens n'est jamais perdu non plus.

   Par contre, les paquets suivants peuvent parfois se perdre. Vous devrez donc gérer cette possibilité.

2. Si un paquet se perd, il est acceptable que le son soit momentanément interrompu chez le client. Par contre, la transmission doit reprendre dès que d'autres paquets sont reçus. Vos programmes ne devront donc pas se bloquer si un paquet est perdu.

   Il n'est pas obligatoire d'implémenter un système de buffering du côté du client pour faire en sorte de supporter des pertes de paquets sans interruption de la sortie audio.

3. Votre serveur peut être conçu pour ne servir qu'un seul client à la fois. Si un autre client se présente alors que le serveur est déjà occupé à transmettre un flux audio à un autre client, alors il peut refuser de servir le nouveau client et lui retourner un message d'erreur.

4. Vos programmes ne doivent pas nécessairement être compatibles avec ceux des autres étudiants (par exemple: il n'est pas nécessaire que votre client fonctionne avec le serveur d'un autre groupe). Ce sera largement suffisant si vos deux programmes fonctionnent l'un avec l'autre.

Optionnel: concevez votre serveur pour pouvoir servir plusieurs clients simultanément. Attention: cela rajoute un niveau de complexité supplémentaire. Je vous suggère fortement de commencer par un serveur qui ne sert qu'un seul client à la fois, puis de l'étendre pour supporter plusieurs clients simultanés.

Merci d'écrire une petite **documentation de *2 pages maximum*** qui explique comment fonctionne votre protocole, et de l'envoyer à votre encadrant en même temps que vos programmes.

A rendre pour le vendredi 5 avril 2019: filtres audio

Vous devez maintenant étendre vos programmes pour qu'ils permettent de modifier le son des fichiers audio. Vous devez implémenter au moins 3 filtres de votre choix.

L'interface de votre serveur doit rester la même que précédemment. L'interface de votre client doit désormais être la suivante:

```
audioclient server_host_name file_name filter_name [filter_parameter]
```

Votre client doit fonctionner même si aucun filtre n'est demandé. Merci d'étendre vore documentation pour indiquer la liste des filtres que vous avez implémenté, la façon de les utiliser, et l'effet sonore qu'ils sont censés réaliser.

Je vous suggère de commencer avec des filtres très simples, puis si vous le souhaitez d'essayer un ou deux filtres plus complexes. Voici quelques exemples de filtres simples:

1. Transférer un signal mono même si le fichier original est en stéréo
2. Ajuster le volume du son
3. Ajouter de l'écho

Attention: réfléchissez soigneusementà la façon dont vous concevez l'*interface* de chaque filtre et la façon dont il interagira avec le reste du système. Cet aspect est l'une des principales difficultés de cet exercice.

Optionnel: concevez votre système de façon à pouvoir utiliser plusieurs filtres simultanément (par exemple: ajouter de l'écho et ajuster le volume en même temps).

— fin —