ÉCOLE
SUPÉRIEURE
D'ÉLECTRICITÉ

Supélec

Version 011

Tutoriel

Supelec-audio.

Auteur : Jean-Louis Gutzwiller.

Dernière modification : 21 octobre 2013

Jean-Louis.Gutzwiller@supelec.fr

Table des matières

1 Introduction

Supelec-audio vous permet d'utiliser facilement, en langage C ou en langage C++, les systèmes audio de la machine. Cette bibliothèque a été conçue pour s'interfacer sur plusieurs types de matériel et elle est capable de gérer des cartes comportement plus que deux entrées/sorties (au-delà de la simple stéréo).

En outre, le module « supelec-audio-remote » permet d'accéder à la carte son d'une machine distante.

2 Systèmes cibles

Supelec-audio fonctionne sur Windows et sur Linux

- Sur Windows : les programmes peuvent utiliser les fonctions de Windows en natif (limitées à la stéréo pour la plupart des plateformes Windows).
- Sur Linux : les programmes peuvent utiliser le device standard /dev/audio (qui existe sur certaines distributions de Linux, mais n'existe plus sur la distribution Fedora utilisé sur le campus de Metz de Supélec), mais également les nouvelles API proposées par « Alsa », « Pulseaudio » et « Jack-audio ». À noter que les cartes audio disposant de multiples entrées/sorties (au-delà de la stéréo) peuvent être prises en compte par le système « jack-audio » sur Linux.

Dans la plupart des cas, Supelec-audio détermine de lui-même les fonctionnalités à utiliser (par exemple si Jack-audio est disponible sur Linux), mais l'utilisateur peut également choisir d'activer un système audio particulier.

3 Utilisation de Supelec-audio

3.1 Introduction

Vous pouvez vous inspirer des « makefiles » pour comprendre le principe. Le fichier source de votre programme fera un « include » de « supelec-audio2.h » :

```
#include   <supelec-audio2.h>
```

Vous pourrez alors utiliser les fonctions de Supelec-audio de la manière expliquée ci-dessous.

Si vous souhaitez utiliser la version permettant d'accéder à des cartes son d'un ordinateur distant, effectuez :

```
#include   <supelec-audio-remote.h>
```

Notons que la version « remote » autorise l'utilisation conjointe d'une carte son locale et d'une carte son distante.

3.2 Note de version

Depuis la version 1.04 (qui correspond à la mise en œuvre de la version « remote »), les noms des fonctions ont changé. Toutes les fonctions sont préfixées par « supelec-audio_ ». Vous pouvez cependant utiliser les noms courts comme pour les versions précédentes en compilant votre programme avec l'option : « -D_SUPELEC_AUDIO_SHORT_NAMES_ ».

Donc :

- si vous utilisez une ancienne version de « supelec-audio2 » (avant la version 1.04), ou si vous définissez la macro « _SUPELEC_AUDIO_SHORT_NAMES_ », vous utilisez les noms courts, c'est à dire les noms sans le préfixe « supelec_audio_ » ;
- si vous utilisez une version de supelec-audio2 supérieure ou égale à 1.04, et que vous ne définissez pas la macro indiqué ci-dessus, vous utilisez les noms long, c'est à dire commençant par le préfixe « supelec_audio_ »;

3.3 Principe général

3.3.1 Boucle principale

Un programme audio est en principe destiné à fonctionner en temps réel et en permanence, donc à durée illimitée. Le programme est donc basé sur une boucle. Il est possible de prévoir une condition d'arrêt si l'utilisateur émet une action spécifique (par exemple, en cliquant sur un bouton demandant d'arrêter le traitement).

Notez que la boucle audio peut être écrite dans un second thread, indépendant du thread principal de l'application.

3.3.2 Buffers audio

La boucle devra débuter par une récupération des buffers audio. Il y a deux fonctions prévues pour cela :

- récupération du buffer d'entrée (il contient les échantillons acquis par le microphone)
- récupération du buffer de sortie (il est destiné à recevoir les échantillons que votre programme génère, en vue de les faire jouer sur les haut-parleurs).

Bien-entendu, si vous n'utilisez que les entrées ou que les sorties, il n'est pas nécessaire d'appeler la fonction traitant du sens inverse.

La boucle devra terminer par le relachement des buffers (c'est à dire elle doit rendre les buffers qu'elle a précédemment obtenu) :

- pour l'entrée : le système récupère la mémoire du buffer pour y placer de nouveaux échantillons,
- pour la sortie : le son sera joué par les haut-parleurs.

Les fonctions sont les suivantes :

- Récupération du buffer en entrée :

  ```
  supelec_audio_echantillon * supelec_audio_recupereBufferEntreeEtTaille(int * taille);
  ```

- Récupération du buffer de sortie :

  supelec_audio_echantillon * supelec_audio_recupereBufferSortieEtTaille(int * taille);

Notons que ces deux fonctions renvoient un pointeur sur le buffer récupéré. Attention : ce pointeur peut changer (ou non) d'un appel à l'autre. Les fonctions mettent également à jour la variable « taille » dont la référence doit être passée en paramètre. Cette taille indique le nombre d'échantillons par voie dans le buffer. Cette taille est forcément la même en entrée et en sortie (les deux fonctions renvoient donc la même valeur).

- Relâchement du buffer d'entrée :

  ```
  void supelec_audio_libereBufferEntree();
  ```

- Relâchement du buffer de sortie :

  ```
  void supelec_audio_valideBufferSortie();
  ```

Le type « supelec_audio_echantillon » est déclaré d'un type flottant disponible sur la machine. Les échantillons sont des valeurs comprises entre -1 et 1. Vous pouvez toujours copier une variable de type « supelec_audio_echantillon » vers une variable d'un type flottant de la machine (float ou double) et réciproquement.

La version actuelle de « supelec-audio2 » définit explicitement le type « supelec_audio_echantillon » comme étant un float.

3.3.3 Initialisation

Le système audio est initialisé par l'appel de l'une des deux fonctions suivantes :

- `int supelec_audio_initialiseSonCanaux(int frequence, int canauxIn, int canauxOut, int tailleBuffer);`

  Cette fonction initialise le système audio en demandant une fréquence d'échantillonnage (qui sera la même en entrée et en sortie). On spécifiera également le nombre de canaux (1 = mono, 2 = stéréo, ...) en entrée et en sortie. On pourra également demander une taille de

buffer particulière.
Si la fréquence d'échantillonnage demandée est nulle, alors le système utilise une fréquence par défaut.
La taille du buffer demandée n'est pas forcément respectée par le système. Chaque système fournira une taille de buffer comme il le pourra, en respectant, seulement si cela est possible, la taille demandée. Cette taille de buffer est exprimée en nombre d'échantillons par voie.
Le nombre de canaux demandé sera respecté, mais le fonctionnement exacte dépend du système. Pour certains systèmes stéréo, demander un seul canal en sortie fera jouer le son sur les deux haut-parleurs simultanément (son mono sur un système stéréo), alors que d'autre systèmes ne feront jouer le son que sur l'un des deux haut-parleurs.

En cas de succès, la fonction renvoie la fréquence d'échantillonnage. En cas d'échec, la fonction renvoie 0.

- `int supelec_audio_initialiseSonCanauxParNom(int frequence, int canauxIn, int canauxOut, int tailleBuffer, char * name);`

  Cette fonction est analogue à la précédente, mais elle permet en plus de spécifier le système audio qu'on utilise par un nom. Le nom peut être :

  → « winaudio » : utilisation du système natif Windows

  → « winaudio2 » : utilisation du système natif Windows en 32 bits

  → « devaudio » : utilisation du système natif /dev/audio sur Linux

  → « pulseaudio » : utilisation du système pulseaudio sur Linux

  → « alsa » : utilisation du système alsa sur Linux

  → « jack » : utilisation du système jack-audio sur Linux

  Le nom peut également représenter un périphérique :

  → « winaudio:in=1,out=2 » : pour sélectionner un device sous Windows (entrée et sortie)

  → « /dev/audio », « /dev/dsp » : sur Linux

  → « hw:0.0 » : identifie un périphérique particulier d'alsa

  → « hw:0,0;hw1,0 » pour identifier un périphérique d'entrée et un périphérique de sortie pour alsa

  → « jack:nom » : utilise le système jack, mais en enregistrant le programme sous le nom indiqué

3.3.4 Fermeture

Lorsque l'audio n'est plus nécessaire, par exemple lorsque le programme se termine, il convient de fermer le périphérique audio en appelant la fonction suivante :

- `int supelec_audio_termineSon(int attente);`

  Cette fonction stoppe le système audio. Le paramètre « attente » est prévu, pour une version future de sorte que :

  --> attente = 1 : les buffers audio en sortie qui ont été validés seront entendus avant que le système ne s'arrête,

  --> attente = 0 : l'arrêt est immédiat.

Dans la version actuelle, ce paramètre n'est pas pris en compte et le comportement est non spécifié.

Remarque importante : vous devez absolument fermer le système audio en quittant votre programme, sous peine de ne plus pouvoir le redémarrer. Sur certains systèmes, en effet, l'arrêt d'un programme qui ne donne pas lieu à fermeture propre du système audio bloque la carte son.

3.4 Les fonctions pour la version utilisant uniquement la carte locale

La version de « supelec-audio » utilisant la carte locale s'appelle « supelec-audio2 ».

Pour l'utiliser, effectuez :

```
#include <supelec-audio2.h>
```

Vous disposez alors des fonctions suivantes :

3.4.1 Version avec les noms courts

Vous utilisez les noms courts si vous êtes dans l'une des deux conditions ci-dessous :

- la version de « supelec-audio2 » est antérieure à la version 1.04
- vous avez définie l'option de compilation suivante sur la ligne de commande du compilateur : -D_SUPELEC_AUDIO_SHORT_NAMES_

Les fonctions sont les suivantes :

```
int initialiseSonCanaux(int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer);
int initialiseSonCanauxDriver(int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer, int driver, const char * name);
int initialiseSonCanauxParNom(int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer, const char * name);
int termineSon(int attente);
echantillon * recupereBufferSortieEtTaille(int * taille);
void valideBufferSortie();
echantillon * recupereBufferEntreeEtTaille(int * taille);
void libereBufferEntree();
unsigned long int recupereTailleBufferEntree(void);
unsigned long int recupereTailleBufferSortie(void);
```

Ces fonctions réalisent les opérations suivantes :

- ***initialiseSonCanaux*** : initialise le système audio local en indiquant la fréquence d'échantillonnage souhaitée, le nombre de canaux en entrée et en sortie et la taille du buffer de données (en nombre d'échantillons par canal). Notons que la carte son ou le système audio sous-jacent peut éventuellement refuser les paramètres fournis. Vous devez donc fournir des paramètres en adéquation avec le système audio utilisé. Exception : si le système audio refuse la taille de buffer demandée, il peut en fournir une autre. Les fonctions d'échange de données renvoient la taille effective du buffer.

Note : le système tente de trouver automatiquement un driver informatique disponible pour utiliser la carte son. Cela peut éventuellement échouer, auquel cas il faudra utiliser l'une des deux fonctions suivantes pour spécifier explicitement le driver à utiliser.

- ***InitialiseSonCanauxDriver*** : comme la précédente, mais permet en outre de spécifié un driver informatique (logiciel) à utiliser. Les drivers sont à choisir dans une liste de constantes : DRIVER_AUTO (choix automatique, comme la fonction précédente), DRIVER_WINAUDIO (driver natif de Windows, ne fonctionne bien évidemment que sur ce système), DRIVER_DEVAUDIO, DRIVER_ALSA, DRIVER_OSS, DRIVER_JACK, DRIVERPULSEAUDIO (ces derniers fonctionnent sur Linux, à l'exception du driver oss qui n'a pas été implémenté et ne peut donc pas être utilisé).
  Dans le cas d'un choix de driver automatique, le driver JACK sera tenté en premier. Si un serveur jackd tourne sur la machine, ce dernier sera utilisé. Dans le cas contraire, la fonction se replie sur ALSA ou PULSEAUDIO.
  Le paramètre « driver » permet de spécifier l'une de ces constantes.
  Le paramètre « name » prendra le nom physique du device à utiliser. Par exemple, pour le deriver DEVAUDIO, les noms sont des devices tels que /dev/audio sur Linux. Si plusieurs cartes son sont disponibles, il faudra choisir la bonne.
- ***initialiseSonCanauxParNom*** : comme la précédente, mais le type de driver est déterminé automatiquement par le nom. Ainsi, le paramètre « name » pourra prendre comme valeur :
  → « /dev/audio », « /dev/audio2 »... : indique d'utiliser le driver DRIVER_DEVAUDIO
  → « alsa », « hw:0,0 », … : indique d'utiliser le driver DRIVER_ALSA (les noms en « hw:0,0 » sont des noms pour identifier les cartes son sous Alsa)
  → « pulseaudio » : indique d'utiliser le driver DRIVER_PULSEAUDIO ; dans ce cas il n'y a pas de mécanisme pour sélectionner la carte son.
  → « jack » : indique d'utiliser le serveur jackd.
- ***termineSon*** : permet d'arrêter l'utilisation de la carte son.
- ***recupereBufferSortieEtTaille*** : permet d'obtenir un buffer mémoire dans lequel notre programme placera les échantillons à faire entendre sur les haut-parleurs (donc en sortie). La fonction demande un paramètre recevant la taille (en nombre d'échantillons par canal). La taille de ce tableau est donc de ce nombre d'échantillons multiplié par le nombre de canaux demandé en sortie. Le programme devra donc remplir ce tableau avec les échantillons qu'il souhaite faire entendre, en les plaçant en mode entrelacé : on placera en premier les *N* échantillons correspondant au premier instant des *N* canaux en sortie, puis les *N* échantillons correspondant aux deuxième instant des *N* canaux de sortie et ainsi de suite.
  Lorsque cela aura été fait, le programme devra appeler la fonction « valideBufferSortie ».
- ***valideBufferSortie*** : transmet les échantillons ci-dessus à la carte son.
- ***recupereBufferEntreeEtTaille*** : récupère un buffer en entrée qui contient donc les échantillons ayant été acquis depuis l'entrée audio (microphones).
  Les échantillons sont placés en position entrelacés (comme pour la fonction de sortie ci-dessus). Attention au fait que le nombre de canaux en entrée pourrait être différent du nombre de canaux en sortie (selon les paramètres que vous avez passés à la fonction d'ouverture).
  Votre programme pourra alors consulter les échantillons reçus et effectuer un traitement. Lorsqu'il n'a plus besoin de ces échantillons, il appellera la fonction ci-dessous.
- ***libereBufferEntree*** : indique que notre programme n'a plus besoin du tableau d'échantillons récupéré en entrée par la fonction précédente. Notons que cette libération est obligatoire.
- ***recupereTailleBufferEntree*** : disponible à partir de la version 1.05, cette fonction permet de

connaître la taille du buffer d'entrée sans être obligé de provoquer une première acquisition. La taille de buffer doit se comprendre comme le nombre d'échantillons pour une voie. La taille effective du buffer est de fait égale à la valeur renvoyée par cette fonction multipliée par le nombre de voies demandé en entrée.

- ***recupereTailleBufferSortie*** : disponible à partir de la version 1.05, cette fonction permet de connaître la taille du buffer de sortie sans être obligé d'acquérir un buffer.
  La taille de buffer doit se comprendre comme le nombre d'échantillons pour une voie. La taille effective du buffer est de fait égale à la valeur renvoyée par cette fonction multipliée par le nombre de voies demandé en sortie.

Note : les fonctions donnant l'information de taille des buffers sont séparées sémantiquement pour les entrées et les sorties. Cela sous-entend que les tailles des buffers en entrée ou en sortie pourraient être différentes. La version actuelle de Supelec-audio2 garantit cependant que les tailles de buffers (c'est à dire le nombre d'échantillons pour chacune des voies) sont bien identiques en entrée et en sortie.

3.4.2 Version avec les noms longs

Vous utilisez les noms long si vous êtes dans la situation correspondant aux deux conditions ci-dessous (simultanément) :

- la version de supélec-audio2 est supérieure ou égale à 1.04
- ET vous n'avez pas mis l'option suivante dans la ligne de commande du compilateur : -D_SUPELEC_AUDIO_SHORT_NAMES_

Les noms longs remplacent alors les noms courts vu ci-dessus, tout en effectuant strictement les mêmes opérations.

```
int supelec_audio_initialiseSonCanaux(int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer);

int supelec_audio_initialiseSonCanauxDriver(int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer, int driver, const char * name);

int supelec_audio_initialiseSonCanauxParNom(int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer, const char * name);

int supelec_audio_termineSon(int attente);

supelec_audio_echantillon * supelec_audio_recupereBufferSortieEtTaille(int * taille);

void supelec_audio_valideBufferSortie();

supelec_audio_echantillon * supelec_audio_recupereBufferEntreeEtTaille(int * taille);

void supelec_audio_libereBufferEntree();

unsigned long int supelec_audio_recupereTailleBufferEntree(void);

unsigned long int supelec_audio_recupereTailleBufferSortie(void);
```

3.5 Les fonctions pour la version utilisant une carte distante et/ou la carte locale

3.5.1 Introduction

Cette nouvelle version, sous le nom « supelec-audio-remote » permet de prendre en charge à la fois la carte son local et une carte son d'un PC distant. Il est possible de mélanger les deux (par exemple, de récupérer le son depuis les microphones d'une carte distante, et de faire jouer le son sur la carte locale).

Pour utiliser ces fonctions, effectuez :

```
#include <supelec-audio-remote.h>
```

Remarque importante : si vous utilisez « supelec-audio-remote », n'effectuez pas d'appel à des fonctions de « supelec-audio2 », bien que ces dernières soient disponibles. « supelec-audio-remote » fera en sorte d'effectuer les appels à « supelec-audio2 » pour la carte locale lorsque cela est nécessaire.

3.5.2 Fonctions à disposition

Les fonctions de la version remote :

- n'existe qu'en version longue, c'est à dire avec le prefixe « supelec_audio_ » dans le nom ;
- ont en plus le suffixe « _remote » dans le nom.

Ces fonctions sont les mêmes que pour « supelec-audio2 » et effectuent exactement les mêmes opérations que « supelec-audio2 » si elles sont appelées sur une carte locale. Les fonctions d'ouverture prennent cependant des paramètres supplémentaires permettant de rediriger l'un ou l'autre des flux (ou les deux) vers un ordinateur distant.

Initialisation :

```
    int supelec_audio_initialiseSonCanaux_remote(const char * in, const char * out, int
frequenceEch, int canauxIn, int canauxOut, int tailleBuffer);

    int supelec_audio_initialiseSonCanauxDriver_remote(const char * in, const char * out,
int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer, int driver, const char *
name);

    int supelec_audio_initialiseSonCanauxParNom_remote(const char * in, const char * out,
int frequenceEch, int canauxIn, int canauxOut, int tailleBuffer, const char * name);
```

Ces trois fonctions permettent d'initialiser le système audio de la même manière que les fonctions correspondantes de « supelec-audio2 ». Les deux premiers paramètres (« in » et « out ») ont la signification suivante :

- Si NULL, ou si égal à la chaîne vide (""), indiquent d'utiliser la carte son locale dans le sens correspondant (entrée ou sortie). L'appel de l'une de ces fonctions avec ces deux paramètres NULL ou vide est donc équivalent à appeler la fonction correspondante de « supelec-audio2 ».

- Si une autre chaîne de caractère que les cas ci-dessus est passée en paramètre pour « in » ou pour « out », alors la direction concernée sera connectée à une carte son distante d'une autre machine sur laquelle tourne le serveur « supelec-audio-serveur ».
  Dans ce cas, la syntaxe du paramètre est :

  ```
  nom:port
  ```

  Si le port n'est pas spécifié, le port 5001 est utilisé par défaut (pour cela mettre uniquement le nom de la machine sans les deux-points).

Les autres fonctions sont :

```
int supelec_audio_termineSon_remote(int attente);
supelec_audio_echantillon * supelec_audio_recupereBufferSortieEtTaille_remote(int * taille);
void supelec_audio_valideBufferSortie_remote();
supelec_audio_echantillon * supelec_audio_recupereBufferEntreeEtTaille_remote(int * taille);
void supelec_audio_libereBufferEntree_remote();
unsigned long int supelec_audio_recupereTailleBufferEntree_remote();
unsigned long int supelec_audio_recupereTailleBufferSortie_remote();
```

Elles s'utilisent exactement comme les fonctions équivalentes de « supelec-audio2 ».

Remarque : la liaison ne pourra s'établir que si les paramètres sont cohérents. En particulier :

- La carte local et la carte distante doivent fonctionner sur la même fréquence d'échantillonnage (pour le distant, la fréquence d'échantillonnage est déterminée au lancement du serveur).
- Les tailles de buffers sont garanties identiques dans les deux sens (supelec-audio-remote ajuste la taille du buffer du distant à la taille de buffer du local dans le cas où le local impose une taille de buffer différente de celle demandée) ;
  remarquons à ce propos que la taille de buffer spécifiée pour le client (la machine locale) peut être différente de celle spécifiée au lancement du serveur sur la machine distante.
- Le nombre de canaux sur la carte distante doit être égal au nombre de canaux spécifié au lancement du serveur (ou doit être nul).

3.6 Compilation et édition des liens

3.6.1 Si vous utilisez la carte locale uniquement

Vous pouvez vous inspirer des « makefiles » pour réaliser la compilation.

Exemple de commande de compilation (à placer sur une seule ligne) :

```
gcc mon_programme.c -o mon_programme `pkg-config --cflags
     --libs supelec-audio2`
```

Cette commande compile le programme indiqué quelle que soit la plateforme. Notez que cette commande ne fonctionne que depuis un shell, mais pas depuis une invite de commande Windows. Sous Windows, il sera nécessaire de créer un fichier « makefile » pour que la commande soit correctement prise en compte.

3.6.2 Si vous utilisez la carte locale et/ou une carte distante

Exemple de commande de compilation (à placer sur une seule ligne) :

```
gcc mon_programme.c -o mon_programme `pkg-config --cflags
     --libs supelec-audio-remote`
```

Cette commande compile le programme indiqué quelle que soit la plateforme. Notez que cette commande ne fonctionne que depuis un shell, mais pas depuis une invite de commande Windows. Sous Windows, il sera nécessaire de créer un fichier « makefile » pour que la commande soit correctement prise en compte.

4 Exemples de programmes

4.1 Téléchargement

Vous pouvez télécharger les exemples en suivant le lien :

http://www.metz.supelec.fr/metz/personnel/gutzwiller/_TUTOS_98763456_/20100908/Supelec-audio/supelec-audio-test.zip

Note : ces exemples sont plus récents que ceux listés ci-dessous.

4.2 testaudio

Ce programme récupère l'entrée audio et copie le son vers la sortie audio. Il demande l'ouverture d'un canal mono en entrée et d'une sortie en stéréo. Le même signal est émis sur les deux voies en sortie.
Notons l'utilisation des fonctionnalités de « console.h » qui permet d'intercepter la frappe de CTRL+C au clavier et de provoquer l'arrêt propre du programme.

Utilisation :

testaudio : affiche le mode d'emploi

testaudio <volume> [frequence] [driver]