Ecole des Mines de Saint Etienne

Projet PSE : Rapport
Bibliothèque d'accords

Auteurs :
Kévin Guilloux
Némo Kardassevitch

10 juin 2024

Table des matières

Chapitre 1

Interface Client/Serveur

Cette partie se focalise sur l'explication des interactions entre les codes du client et du serveur.

1.1 La connexion

1.1.1 Architecture

Afin de supporter des connexion multiples, le serveur est construit sur une architecture à workers dynamiques. On affecte ici un worker à chaque client, avec un nombre maximum de workers disponibles. Chaque worker correspond à un thread, qui est ouvert au moment de la connexion du client, et fermé à sa déconnection, libérant le worker pour un potentiel client en attente de connexion.
Ici, le serveur est configuré pour un maximum de 5 workers simultanés, modifiable via une constante située au début du code contenu dans le fichier "*serveur.c*".

1.1.2 Identifiant utilisateur

Lorsqu'un client est connecté, il fait face à une interface de connexion, entèrement hébergée sur le serveur. C'est une constante dans l'ensemble des 2 codes ici présent, le client ne possède que des fonctions d'affichage dans le terminal, et de réception de données utilisées instantanément. Aucun calcul, ou stockage n'est effectué coté client.
L'interface de connexion permet donc de rentrer tout d'abord l'identifiant utilisateur, qui, si il correspond à un utilisateur existant, sera comparé avec une liste contenant l'ensemble des utilisateurs actuellement connectés au serveur. Si cet utilisateur est actuellement connecté sur le serveur, alors il ne sera pas possible de se connecter, et il sera demandé au client d'utiliser un autre compte. Si cet utilisateur n'est en revanche pas connecté, il sera ajouté à la liste des utilisateurs connectés et sera redirigé vers l'interface de saisie du mot de passe.

1.1.3 Mot de passe utilisateur

Une fois arrivé sur l'interface de saisie du mot de passe, l'utilisateur possède 3 tentatives (nombre configurable via une constante) afin de rentrer le bon mot de passe. Si il échoue, le client sera déconnecté de force du serveur. Tout comme l'identifiant, le mot de passe est stocké dans un fichier sur le serveur et les 2 sont liés. Les fonctions décrites dans le chapitre 2 sont ensuite utilisés afin d'extraire les données stockés dans ce fichier.

1.2 Les différents menus de sélection

La bibliothèque d'accords possède plusieurs choix successifs à effectuer afin d'accéder aux accords correspondant à l'instrument possédé, et dont on veut consulter les accords, ou les enregistrer. Il est donc nécessaire de proposer au client une interface claire pour effectuer des choix cohérents.
La solution retenue est celle de la saisie de caractères dans le terminal du client, avec un affichage par section afin de donner la possibilité au client de connaitre toutes les options qui s'offrent à lui, dès qu'il en possède. Les données sont transférés du client vers le serveur par l'intermédiaire d'un tube nommée, mais ne sont pas cryptés. Ci-dessous, voici un exemple de menu que l'on peut retrouver dans cette bibliothèque.

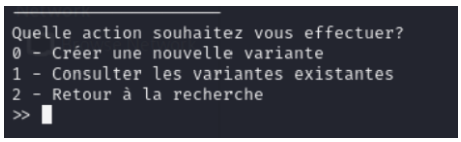

FIGURE 1.1 – Exemple d'un menu de sélection

Des "gardes-fous" sont mis en place afin de sécuriser la sélection dans les menus. Si une commande inexistante est utilisée par le client, il peut, ne rien se passer, retourner au menu de sélection précédent ou encore demander à l'utilisateur de vérifier son écriture (par exemple pour la sélection des profils).

1.3 Les fonctions utilisés

Les fonctions utilisés proviennent de différentes sources :

— La librairie "*pse.h*", qui rassemble les différentes fonctions permettant de réaliser l'architecture client/serveur.
— Les fonctions définies dans le fichier "$fonctionsTraitementUtilisateur.h$", décrits dans le Chapitre 2.
— Les fonctions définies dans le fichier "*accords.h*", décrits dans le Chapitre 3.
— Les fonctions définies dans le client, qui ne sont que des fonctions d'affichages, réalisés dans un but de concission pour le code.

Chapitre 2

Fonctionnement des codes contenu dans fonctionsTraitementUtilisateur.c et fonctionsTraitementUtilisateur.h

2.1 Fichier de stockage des ID et MDP utilisateurs

Les identifiants et mot de passe des utilisateurs sont tous stockés dans un fichier texte situé dans le dossier "*codes*" du projet. Ils sont stockés en brut, et organisés de la manière suivante :

IdUtilisateur1 MotDePasseUtilisateur1
IdUtilisateur2 MotDePasseUtilisateur2
...

Les fonctions servent alors respectivement à récupérer les données d'un utilisateur, en vérifiant que celui-ci existe bien dans le fichier. Et à enregistrer un nouvel utilisateur dans le fichier pour la seconde fonction.

2.2 Fichiers de stockage des profils utilisateurs

Les profils sont tous stockés dans le dossier "$Users\_profile$", et ce, pour tout les utilisateurs. Il existe 2 types de fichier textes qui sont trouvables dans ce fichier :

— Les fichiers de la forme "$IdUtilisateur\_Profils.txt$".
— Les fichiers de la forme "$IdUtilisateur\_NomDuProfil.txt$".

Le premier type de fichier contient le nom de tous les profils appartenant à l'utilisateur, afin de les retrouver efficacement et de ne pas risquer d'accéder à un profil d'instrument appartenant à un autre utilisateur. Toutes les fonctions, excepté la fonction "$recupDataProfil$", n'agisse que sur ce type de fichier. Elles permettent de récupérer les différents noms de profils, vérifier si un profil existe bien (ou déjà dans le cas de la création d'un profil) et enfin, supprimer un profil existant de la liste des profils liés à l'utilisateur. Précision cependant, les fichiers de profils supprimés ne sont pas supprimés, seule la référence de ce fichier est supprimé, il reste donc actuellement stocké dans le serveur afin de pouvoir être récupéré en cas de problèmes.

Le second type de fichier contient quand à lui les caractéristiques propres à chaque profil. C'est à dire, le nombre de cordes, de frettes et enfin le tuning de l'instrument concerné. La majorité des interactions avec ce fichier étant uniquement effectué pendant la création de celui-ci, elle n'est pas rassemblée dans une fonction mais est effectuée directement dans le code du serveur.

2.3 Mémorisation des utilisateurs connectés

Afin de garder en mémoire les utilisateurs connectés, et donc ne permettre qu'une connexion au serveur par utilisateur, pour ne eviter une surchage du serveur par un seul utilisateur qui se connecterait via plusieurs clients par exemple, on se repose sur un stockage global des utilisateurs connectés au serveur. Afin d'adresser le tableau des utilisateurs connectés, qui est une variable globale au niveau du serveur, on a créer les fonctions se terminant

par "*ConnectedUser*". Elles permettent respectivement de vérifier si un utilisateur est déjà connecté, initialiser le tableau, ajouter un utilisateur qui vient de se connecter, retirer un utilisateur qui se serait déconnecté.

Chapitre 3

Fonctionnement des codes contenu dans accords.c et accords.h

3.1 Structure

Les codes des fichiers accords reposent sur quatre structures : Instruments, Accords, Accès et Représentation.
La strucutre Instruments contient l'ensemble des données contenues dans le profil de l'utilisateur, à savoir le nombre de cordes, de frettes, et l'accordage de l'instrument (ou tuning).
La structure Accords contient elle les données spécifique de l'accord et des variantes auxquels on souhaite accéder, c'est-à-dire le nom de l'accord, la première frette sur laquelle l'accord est joué ainsi que le nombre de frettes nécéssaire pour jouer l'accord.
La structure Accès permet de retrouver le chemin d'accès aux fichiers descriptifs des accords. Voici le chemin d'accès générique permettant de retrouver un fichier d'accord : nombreDeCordes_tuning/nomAccord/tailleAccord_premièreFrette
Dans cette strucutre, on stocke donc des chaines de caractère, une correspondant au premier nom de dossier (nombreDeCordes_tuning) et une correspondant au second nom de dossier (nomAccord).
Finalement, la structure Représentation permet de stocker les données utiles à la représentation d'un accord dans une matrice, et garde en mémoire les pointeurs des structures précédentes pour un accord.

3.2 Initialisations

Dans cette section du code sont contenus les codes d'initialisation des quatres structures, ainsi qu'un code permettant une initialisation générale de toutes les structures à partir des données recues lors d'une recherche d'accord.

3.3 Free

Cette section permet de libérer l'ensemble des mallocs fait durant le processus d'initialisation des structures. Encore une fois, on retrouve quatre fonctions pour libérer chacune des strucures ainsi qu'une fonction libérant dans le bon ordre l'ensemble des strucures (Représentation devant être libéré en dernier car seul moyen d'accéder aux trois autres structures).

3.4 Affichages

Les cinq premières fonctions sont similaire au fonctionnement des précédentes : quatre pour afficher les données intéréssantes des structures et un qui regroupe l'affichage de toutes les données.

Un code supplémentaire permet de visualiser une liste des variantes disponibles pour un accord, c'est-à-dire une liste de première frette et de taille d'accord d'un accord donné. Pour ce faire, elle parcourt tous les fichiers contenu dans les dossiers définis par la structure Accès.

3.5 Organisation des dossiers

Ici sont regroupés les codes permettant de travailler sur la création des dossiers et la navigation à l'interieur de ceux-ci.

Les fonctions dossier_Profil et dossier_Accords permettent de créer les dossiers (dans le cas où ils n'existent pas déjà) de la strucure Accès.

La fonction fichier_Existe permet de vérifier si un fichier .txt existe déjà. Elle permet d'éviter qu'un utilisateur souhaitant créer un accord en supprime un autre sans l'avoir vu au préalable.

La fonction nb_Variantes retourne le nombre de fichiers contenu dans un dossier, et donc le nombres de variantes d'un accord qui existent.

3.6 Lecture et écriture des fichiers

Cette partie contient simplement les codes écriture et de lecture d'un fichier à partir des données des structures. Un fichier contient la matrice contenu dans représentation, les autres informations utiles sont contenu dans le chemin d'accès au fichier et son nom.

3.7 Choix utilisateurs

Cette partie contient les fonctions où l'utilisateur doit procéder à un choix, comme par exemple lorsqu'il souhaite consulter un accord et dois décider d'une variante parmi celles disponibles grace à la fonction choix_Variante ou le choix de la première frette et de la taille accord pour la création d'un nouvel accord avec la fonction choix_Construction.

3.8 Actions

Cette dernière partie consiste en les trois fonctions qui décident du déroulé d'actions : celle de modifier un accord existant, de créer un nouvel accord ou bien de consulter un accord.

La fonction nouvel_Accord vérifie que l'accord que l'utilisateur souhaite créer n'existe pas déjà, puis lui permet de l'écrire et enregistre le nouvel accord. La fonction consult_Accord permet de consulter la liste des variantes de l'accord recherché, et de visualiser ensuite la variante désiré. La fonction modifie_Accord est elle utile pour l'écriture d'un accord. Elle passe chacune des frettes en affichant la visualisation de l'accord, et en demandant si cette frette doit etre jouée ou non. Si l'accord existe, on écrit petit à petit par dessus l'ancien accord. Si il est nouveau, on écrit sur un accord vierge.