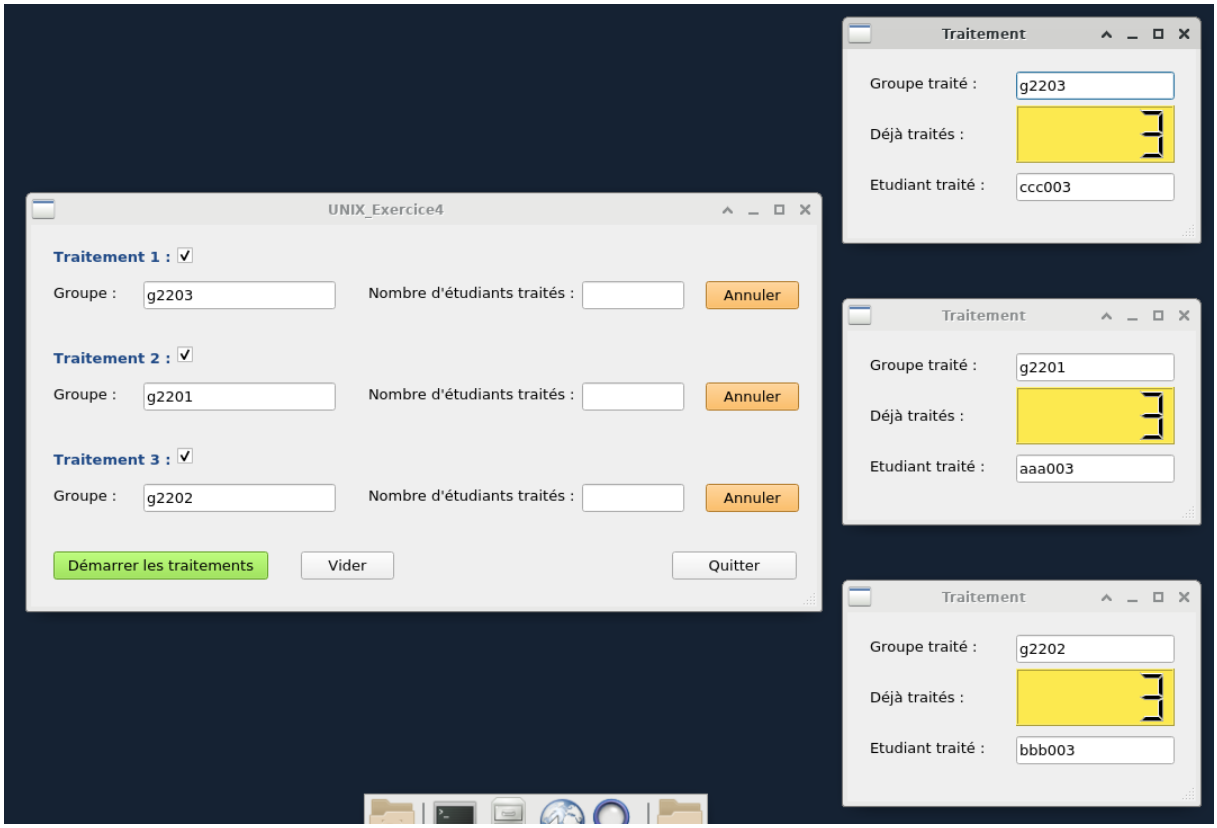
Exercice n°4 : les signaux

Objectifs :

- Se familiariser avec la création de processus (**fork**, **exec**, **exit** et **wait**).
- Se familiariser avec l'envoi et l'armement des signaux (**kill** et **sigaction**).
- Et toujours : création d'un makefile et d'un fichier de traces, se familiariser avec la librairie C d'accès à MySql.

Description générale :

L'application ressemble visuellement à



Le principe général reste le même que dans l'exercice 3. Elle permet de faire une ou plusieurs recherches en base de données et d'y récupérer le nombre d'étudiants du ou des groupes dont on précise le nom dans la partie de gauche de la fenêtre. Les résultats de la recherche s'affichent dans la partie droite de la fenêtre. Trois boutons supplémentaires (« Annuler ») vont permettre d'interrompre à tout moment les processus en cours de traitement.

Lien GitHub : https://github.com/hepl-dsoo/LaboUnix2022_Exercice4

Etape 1 : Création d'un fichier makefile et analyse de l'exécutable Traitement

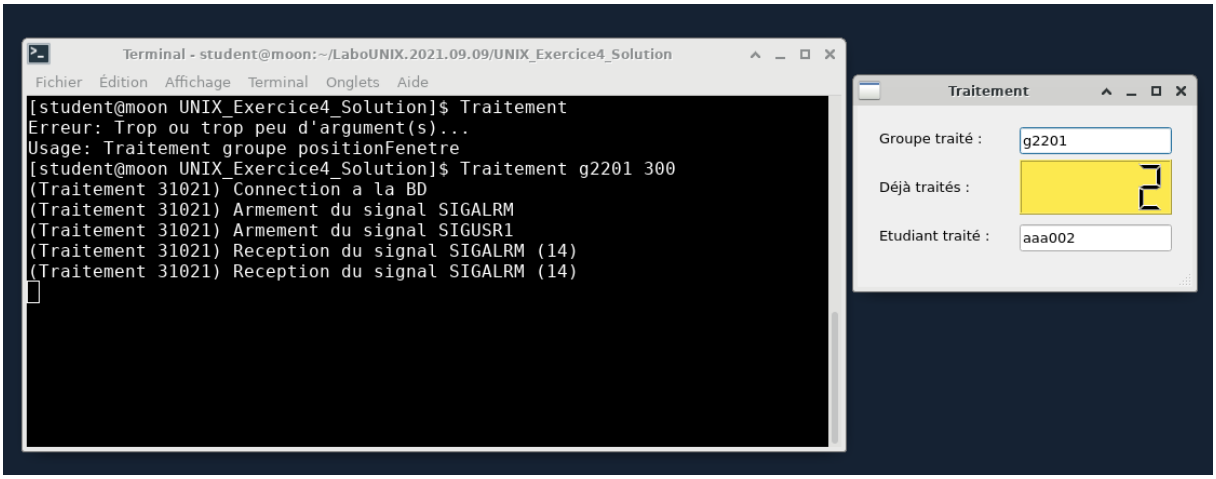
On vous fournit les fichiers suivants :

- **mainex4.cpp, mainwindowex4.cpp, mainwindowex4.h, moc_mainwindowex4.cpp et ui_mainwindowex4.h** : tous les fichiers nécessaires à la création de l'exécutable **UNIX_Exercice4**.

- **maintraitement.cpp, mainwindowtraitement.cpp, mainwindowtraitement.h, moc_mainwindowtraitement.cpp et ui_mainwindowtraitement.h** : tous les fichiers nécessaires à la création de l'exécutable **Traitement**.
- **Compile.sh** : fichier contenant les lignes de compilation nécessaires à la création des fichiers exécutables « **UNIX_Exercice4** » et « **Traitement** ».

Remarquez qu'ici les deux exécutables utilisent la librairie graphique Qt, ils ont donc leurs propres fichiers qu'il faut compiler indépendamment. Le but ici est à nouveau de créer un fichier **makefile** permettant de réaliser la compilation de tous les fichiers nécessaires à la création des exécutables **UNIX_Exercice4** et **Traitement**.

L'exécutable **CreationBD** de l'exercice 3 reste nécessaire pour créer la table UNIX_EX3 qui est encore utilisée dans cet exercice. Le processus **Lecture** (travaillant en console) de l'exercice 3 est ici remplacé par le processus **Traitement** (fenêtre graphique) qui réalise un travail similaire. Celui-ci reçoit toujours en arguments le groupe qu'il faut rechercher (mais également « traiter » ici) ainsi que la position verticale choisie pour la fenêtre :



Plus précisément :

1. Le processus **Traitement** fait un accès à la BD afin de récupérer tous les tuples correspondant au groupe passé en argument (ici g2201) et positionne la fenêtre par rapport au haut de l'écran (ici 300 pixels par rapport au haut de l'écran).
2. Le signal SIGALRM est déjà armé dans ce processus. Grâce à la fonction **alarm(),** le processus entre toutes les secondes dans le handlerSIGALRM dans lequel il affiche dans la fenêtre le nom de l'étudiant traité (le traitement consiste simplement à l'afficher dans la fenêtre) et incrémente de 1 le compteur d'étudiants déjà traités. A chaque réception de SIGALRM, le processus entre donc dans ce handler.
3. Une fois que tous les tuples correspondant ont été traités, le processus retourne, comme le processus Lecture de l'exercice 3, le nombre total d'étudiants du groupe correspondant (utilisation de **exit**).

Le programme **Traitement** est dès lors déjà tout à fait fonctionnel.

Etape 2 : Creation de la table UNIX_EX3 et accès à MySql

Voir étape 2 de l'exercice 3. La table UNIX_EX3 doit exister pour que le processus **Traitement** fonctionne.

Etape 3 : Création asynchrone des processus fils et armement de SIGCHLD

Le principe reste le même que dans l'exercice 3. L'appui sur le bouton « Démarrer les traitements » va provoquer la création de 0, 1, 2 ou 3 processus fils selon que les checkbox sont cochés ou non. De nouveau, pour savoir si un checkbox a été coché par l'utilisateur, vous pouvez utiliser les méthodes

- **bool MainWindowEx4::traitement1Selectionne()**
- **bool MainWindowEx4::traitement2Selectionne()**
- **bool MainWindowEx4::traitement3Selectionne()**

qui retournent true si le checkbox correspondant est sélectionné et false sinon.

L'appui sur ce bouton provoque donc la création des processus fils par appel des fonctions **fork** et **exec** mais la grosse différence par rapport à l'exercice 3 est que **le père ne va pas attendre la fin de ses fils**. Il **n'appelle donc pas la fonction wait** dans la fonction on_pushButtonDemarrerTraitements_clicked() de la fenêtre. Il s'agit donc bien d'un lancement asynchrone des processus fils. Afin que les fenêtres des processus fils ne se chevauchent pas, les 3 fils seront lancés avec des positions verticales de la fenêtre égales à **200**, **450** et **700**.

Le père sera donc prévenu de la fin d'un de ses fils par la réception du signal **SIGCHLD**. Vous devez donc **armer** (utilisation de **sigaction**) le processus père sur le signal SIGCHLD de telle sorte que le père

1. récupère la valeur de l'exit du fils terminé (utilisation de **wait**),
2. supprime ce fils de la table des processus,
3. affiche le résultat du traitement du fils (sa valeur d'exit donc) dans la partie droite de la fenêtre.

Etape 4 : Annulation d'un des processus fils

L'appui sur un des boutons « Annuler » de droite devra provoquer l'arrêt immédiat du fils correspondant qui devra transmettre à son père le nombre d'étudiants qu'il aura déjà traités à ce moment là (la valeur du compteur donc).

Pour cela, l'appui sur le bouton « Annuler » devra provoquer l'envoi du signal SIGUSR1 au processus fils correspondant (utilisation de **kill**).

En vue de réagir correctement à la réception du signal SIGUSR1, vous devez armer (utilisation de **sigaction**), dans le processus Traitement, le signal SIGUSR1 sur un handler dans lequel le processus Traitement retournera la valeur actuelle du compteur avant de se terminer (utilisation de **exit**).

Etape 5 : Implémentation des boutons restants et création d'un fichier de traces

Comme dans l'exercice 3, on vous demande d'implémenter les boutons restants (« Vider » et « Quitter ») et de rediriger la sortie standard d'erreur (stderr) de tous les processus vers un fichier texte appelé « **Trace.log** » (utilisation de **open** et **dup/dup2**).

Bon travail !