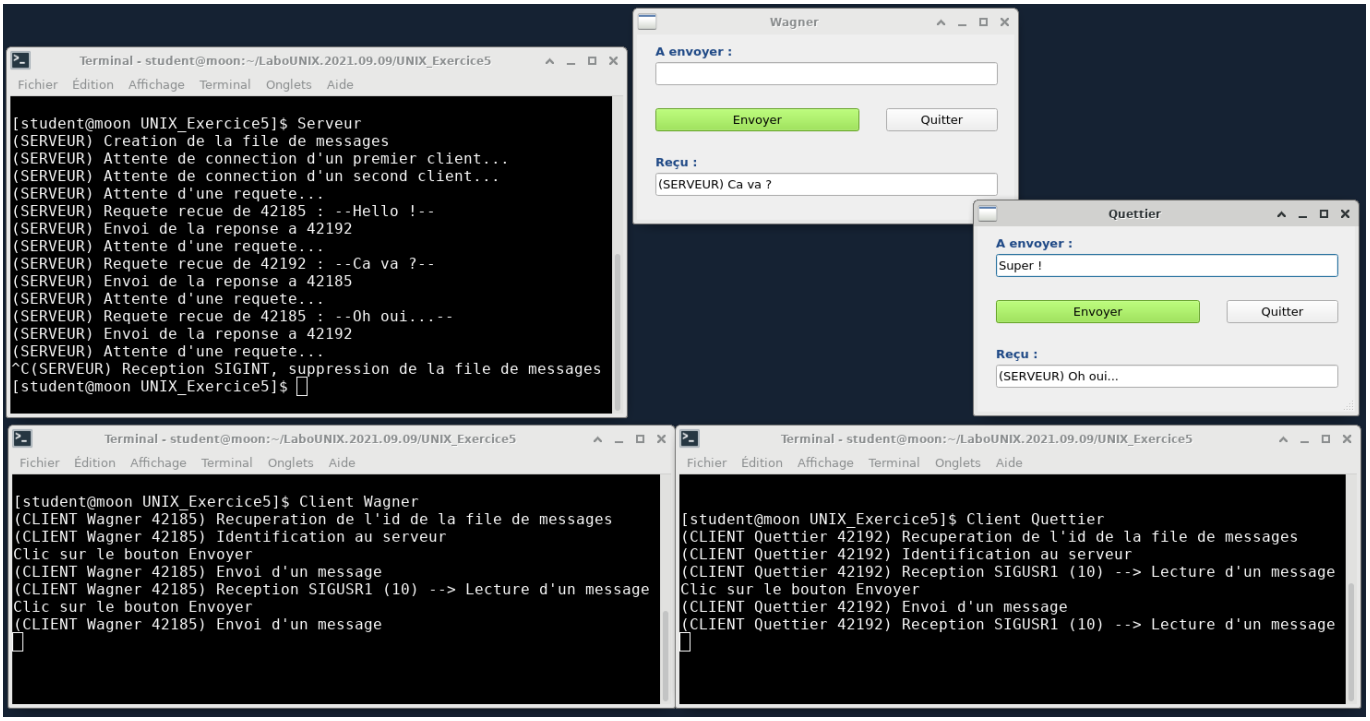
Exercice n°5 : les files de messages

Objectifs :

- Se familiariser avec les files de messages (**msgget**, **msgctl**, **msgsnd** et **msgrcv**).
- Se familiariser avec l'envoi et l'armement des signaux (**kill** et **sigaction**).
- Et toujours : création d'un makefile.

Description générale :

La version finale de l'application ressemble visuellement à



Il s'agit au final de faire communiquer deux applications indépendantes (les « Clients ») par l'intermédiaire d'un processus « Serveur », et cela en utilisant une file de messages.

Lien GitHub : https://github.com/hepl-dsoo/LaboUnix2022_Exercice5

Etape 1 : Création d'un fichier makefile

On vous fournit les fichiers suivants :

- **main.cpp, windowclient.cpp, windowclient.h, moc_windowclient.cpp et ui_windowclient.h** : tous les fichiers nécessaires à la création de l'exécutable **Client**.
- **Serveur.cpp** : Code source de base fournissant l'exécutable **Serveur**.
- **Compile.sh** : fichier contenant les lignes de compilation nécessaires à la création des fichiers exécutables « **Client** » et « **Serveur** ».

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 **Client** et **Serveur**.

Les fichiers à modifier dans la suite seront **Serveur.cpp** et **windowclient.cpp**.

Etape 2 : Creation de la file de messages et premier envoi/réception

Dans cette étape, **le serveur va créer la file de messages** tandis qu'**un seul client est lancé**. Celui-ci va envoyer un message sur la file de messages et le relire lui-même directement.

Vous devez modifier **Serveur.cpp** de telle sorte que

- Le serveur crée la file de messages (utilisation de **msgget**). La clé de la file de messages est définie dans le fichier partagé **protocole.h**.

Vous devez ensuite modifier **windowclient.cpp** de telle sorte que

- Le processus client récupère l'identifiant de la file de messages (utilisation de **msgget**). A nouveau, la clé de la file de messages est définie dans le fichier partagé **protocole.h**.
- Lors du clic sur le bouton « Envoyer », le processus client envoie un message (structure **MESSAGE** définie dans le fichier **protocole.h**, utilisation de **msgsnd**) contenant la chaîne de caractères à envoyer. Une fois envoyé, le processus client lit directement ce message sur la file de messages (utilisation de **msgrcv**) et affiche la chaîne de caractères reçue dans le champ de texte du bas de la fenêtre. Tout se passe donc dans la méthode **void WindowClient::on_pushButtonEnvoyer_clicked()**.

Cette étape sert simplement à mettre en place la file de messages et faire un premier test d'envoi et de réception d'un message.

Etape 3 : Envoi synchrone d'un message par le client

L'idée reste la même que dans l'étape 2, **toujours un seul client**. La différence est que

1. le client envoie le message au serveur.
2. le serveur concatène la chaîne de caractères reçue avec la chaîne « (SERVEUR) ». Par exemple, si le client envoie « Hello ! », le serveur va lui répondre « (SERVEUR) Hello ! ».
3. le serveur envoie la réponse au client.

Vous devez donc modifier **Serveur.cpp** de telle sorte que dans sa boucle principale,

1. le serveur se mette en attente d'un message de **type 1** (convention pour le serveur).
2. le serveur construise un message de réponse dont le **type** est égal au **pid du processus client** et dont le texte est décrit ci-dessus.
3. le serveur envoie la réponse sur la file de messages.

Au niveau du client, tout se passe à nouveau dans la méthode **void WindowClient::on_pushButtonEnvoyer_clicked()** dans laquelle il envoie un message et **attend la réponse du serveur**. Il s'agit donc d'un envoi « **synchronisé** » d'un message (c'est-à-dire qu'il attend la réponse).

Il est important de remarquer ici que le **type d'un message** correspond toujours au **pid du processus destinataire** du message, à l'exception du serveur pour lequel le type est choisi à 1 par convention.

Etape 4 : Envoi asynchrone d'un message par le client

L'envoi du message devient **asynchrone**, c'est-à-dire que dans la méthode **void WindowClient::on_pushButtonEnvoyer_clicked()**, le client envoie le message **mais n'attend pas la réponse du serveur**.

Lorsque le serveur enverra la réponse au client, celui-ci devra être prévenu de manière aysnchrone, c'est-à-dire par l'envoi du signal SIGUSR1 par le serveur (utilisation de **kill**).

Vous devez donc **armer le signal SIGUSR1** (utilisation de **sigaction**) dans le processus client sur un handler dans lequel

1. il lira le message qui lui est destiné sur la file de message,
2. il affichera la réponse reçue dans la fenêtre.

Etape 5 : Deux clients communiquent entre eux

On lancera à présent **2 clients qui vont communiquer entre eux**. **Un message envoyé dans la fenêtre d'un client apparaitra donc dans la fenêtre de l'autre client**.

Pour cela, le serveur doit connaître le pid de chaque client connecté (les variables **pid1** et **pid2** déjà déclarées dans **Serveur.cpp**) afin de rediriger les messages reçus vers le bon client. Dès lors, lors du lancement de chaque processus client, ceux-ci doivent (avant le moindre clic sur le bouton « Envoyer ») envoyer au serveur un **message d'identification** contenant leur pid.

A nouveau, les envois de message sont **asynchrones**. En effet, un client ne peut pas savoir à quelle moment l'autre client va lui répondre. Le serveur doit donc toujours envoyer un signal **SIGUSR1** au client destinataire qui ira lire le message reçu dans le handler de SIGUSR1.

Etape 6 : Suppression propre de la file de messages

Lors d'un **<CTRL-C>** ou l'envoi du signal **SIGINT** au serveur, on aimerait que celui-ci supprime par programmation (utilisation de **msgctl**) la file de messages avant de se terminer.

Vous devez donc armer le signal SIGINT (utilisation de **sigaction**) en conséquence dans le processus serveur.

Bon travail !