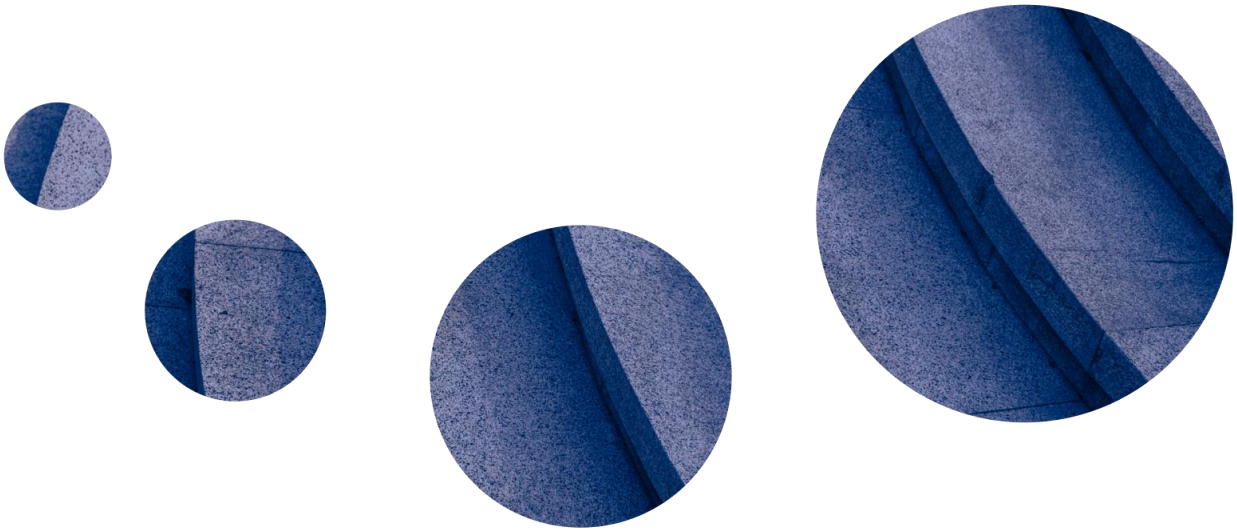


PROJET UNIX

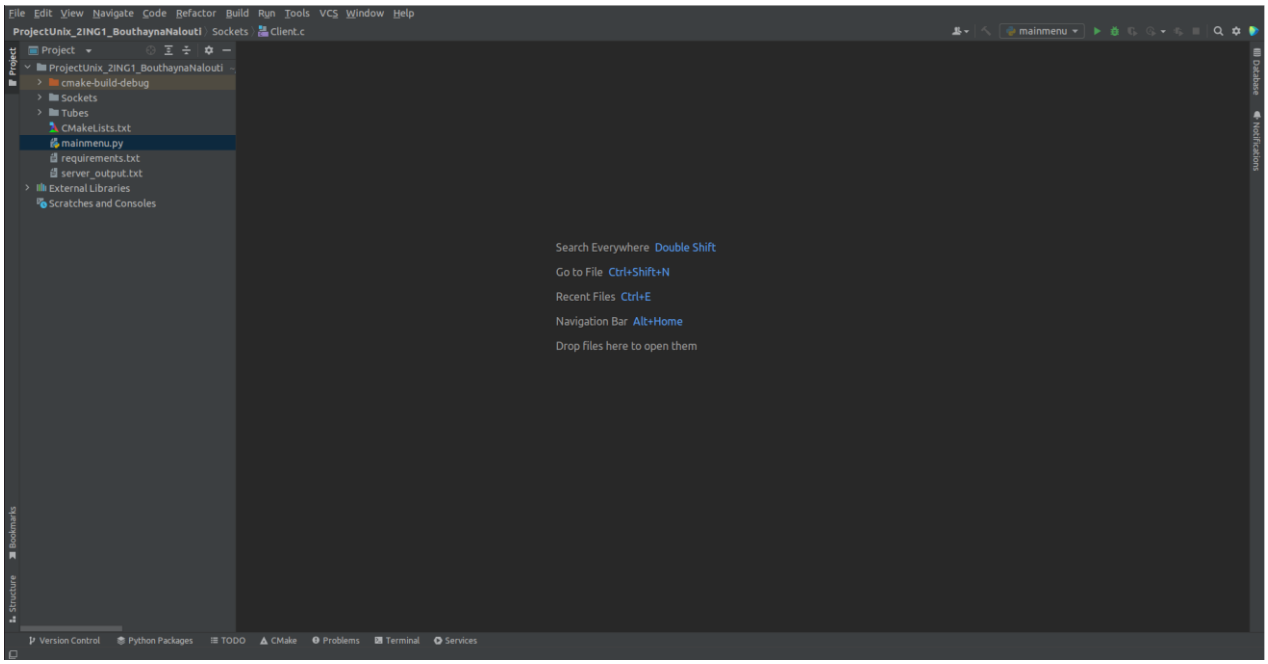
2022-2023

BOUTHAYNA NALOUTI 2èME ING1

Environnement de travail:

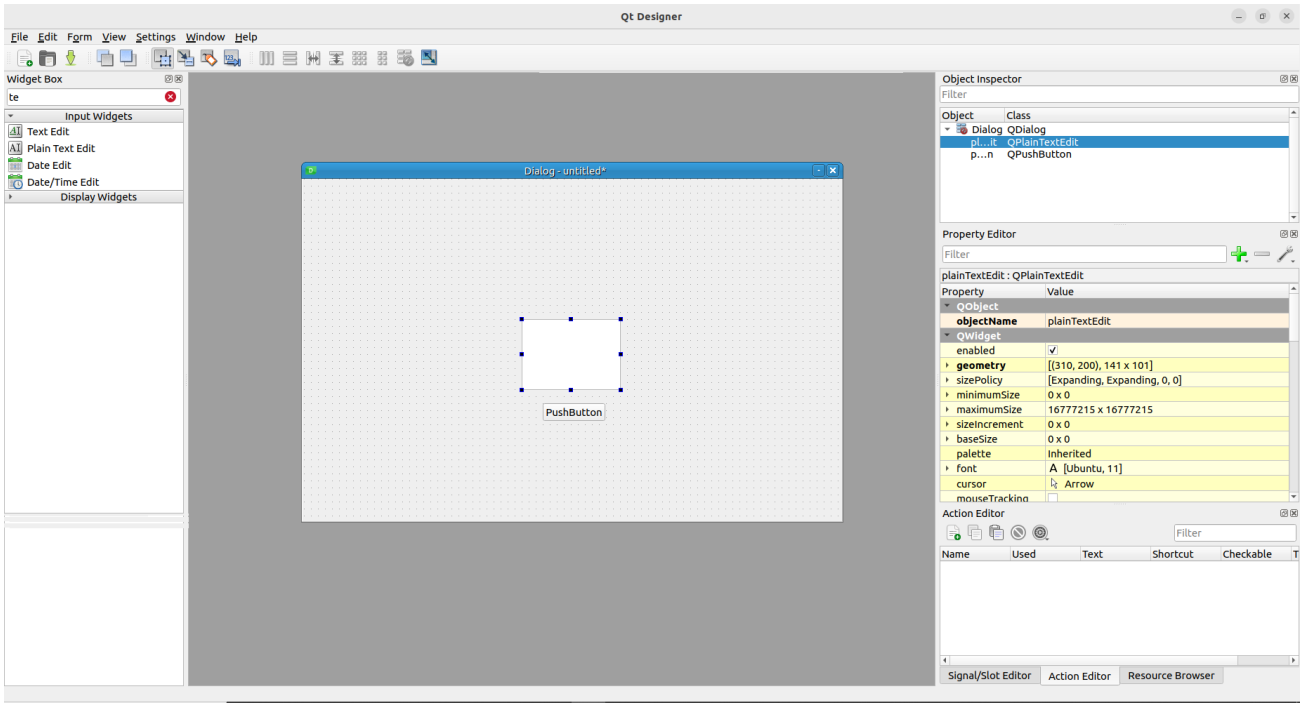
CLion:

CLion est l'IDE que nous avons utilisé pour réaliser ce projet. C'est un IDE extremement puissant facilitant le codage et l'implémentation d'un fichier Python dans un projet C



Qt-Designer:

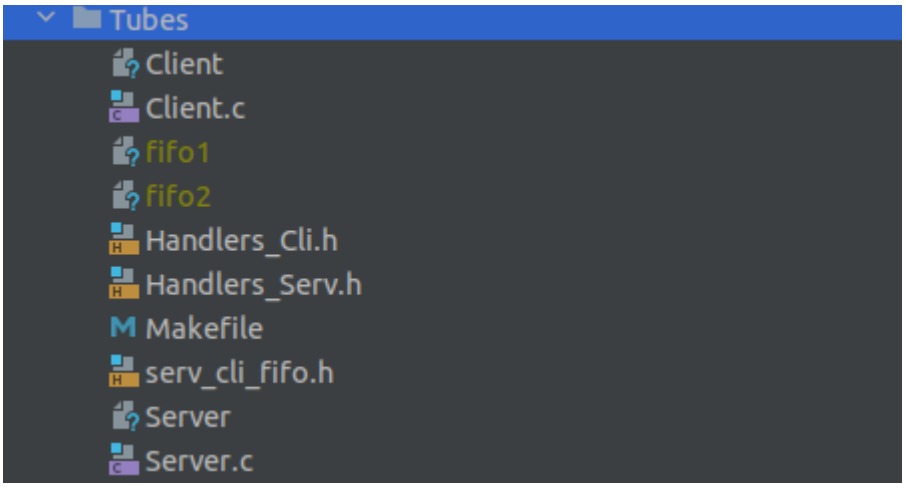
Qt-Designer est l'outil utilisé afin de realiser l'UI demandé, grâce à son drag and drop, il est facile d'implémenter des interfaces assez complexes et par suite d'ajouter des fonctions aux Widgets utilisé



Partie Tubes Nommées :

Architecture :

Nous avons un total de six fichiers enregistrés sous le repertoire “Tubes”. Il y’a trois fichiers Headers, deux fichiers Source et un fichier Makefile qui sont comme suit :



Server.c/Client.c: ces deux fichiers contiennent le code de fonctionnement.

serv_cli_fifo.h: Fichier header qui contient la déclaration et l’importation des bibliothèques nécessaire.

Handlers_Cli.h : fichier header qui contient les fonctions handler.

Handlers_Serv.c: fichier header qui contient les fonctions handler à invoquer côté serveur.

Présentation :

Le principe de notre travail c'est d'avoir un serveur qui est à l'écoute d'un ou plusieurs clients qui vont poser une question. Une question est une demande de générer n entiers aléatoires.

Nous avons deux tubes, Le premier est nommé FIFO1, où les questions seront écrites, le deuxième est nommé FIFO2, où les réponses seront écrites.

```
#define FIFO1 "fifo1"
#define FIFO2 "fifo2"
```

Nous avons crée deux Structures: "quest" et "rps" pour contenir les questions et les réponses

```
struct rps{
    int pid;
    int resultat[NMAX];
};

struct quest{
    int pid;
    int n;
};
```

Format de la question :

Tout d'abord nous enregistrons l'ID du client dans la structure de la question "pid" ainsi que le nombre aléatoire généré dans "n" et finalement on écrit dans le tube nommé

```c
/* Construction et envoi d'une question */
qt.pid = getpid();
srand( seed: getpid());
qt.n = rand() % NMAX + 1;
write( fd: fd1,  buf: &qt,  n: sizeof(qt));
```

Format de la réponse :

La réponse sera stockée dans la structure de la réponse “pid” va contenir l’ID du serveur, cela va nous être utile pour le traitement des signaux plus tard, et “resultat” va contenir les nombres générés aléatoirement

```c
/* construction de la réponse */
rep.pid = getpid();
for(int i=0; i < qt.n; i++){
        rep.resultat[i] = rand() % NMAX + 1;
}
```

Protocole implémenté :

Au lancement du programme, le client génère une question et va se mettre en attente jusqu’à ce qu’il reçoive une réponse, une fois cette dernière est prête, le serveur envoie un signal de type SIGUSR1 au client afin de le réveiller et par suite l’informer que sa réponse est prête. Suite à la réception de sa réponse, il va informer le serveur qu’il a bel et bien reçu sa réponse en lui envoyant lui aussi un signal de type SIGUSR1. Le serveur sera à l’écoute d’autres questions, pour arrêter le serveur, il suffit d’invoquer n’importe quel signal autre que le SIGUSR1

Handler du serveur :

```c
void hand_reveil(int sig){
    printf( format: "Signal %d received\n", sig);
}


void fin_serveur(int sig){
    if(sig != SIGUSR1){ // Any signal except SIGUSR1 will terminate the server
        printf( format: "Signal %d received, terminating server\n", sig);
        exit( status: 0);
    }
}
```

Handler du client :

```
void hand_reveil(int sig){
    printf( format: "Signal %d reçu de la part du serveur\n", sig);
}
```

Partie Sockets :

Architecture :

Nous avons un total de quatres fichiers enregistrés sous le repertoire “Sockets”. Il y’a un fichier Header, deux fichiers Source et un fichier Makefile qui sont comme suit :

Server.c/Client.c: ces deux fichiers contiennent le code de fonctionnement.

Server_Client.h: Fichier header qui contient la déclaration et l’importation des bibliothèques nécessaire.

Présentation :

Le principe de notre travail est le même que dans la partie précédente, les Tubes nommées, sauf qu’ici l’implémentation diffère, elle est beaucoup plus simple comparé

aux tubes vu que les Sockets sont considéré comme étant avancé comparé à ces derniers

Nous avons défini le port que notre serveur Socket va utiliser comme suit :

```
#define PORT 8080
```

Format de la question :

Afin d'envoyer la question, le client utilise la fonction send() qui contient le socket du client en tant que premier paramètre ainsi que le numéro aléatoire généré par le client dans le buffer.

```
/* Construction et envoi d'une question */
send( fd: network_socket,  buf: &num_client,  n: sizeof(num_client),  flags: 0);
printf( format: "Numéros envoyées au total: %d\n", num_client);
```

Format de la réponse :

La réponse sera stockée dans un tableau, les nombres aléatoire seront généré et par suite envoyé au socket Client

```
/* construction de la réponse */
int num_gen[num_client];
for (int i = 0; i < num_client; i++){
    num_gen[i] = rand() % NMAX + 1;
}
```

Protocole implémenté :

Comme mentionné auparavant, l'implémenation des sockets est beaucoup plus simple comparé au Tubes. Au lancement du programme, le client génère une question et l'envoi via "send()", il sera par suite à l'écoute du serveur qui a été déjà initialisé, le client aura par suite sa réponse et sera affiché. Le serveur peut accueillir plus qu'un seul client au même temps vu qu'il est multi-client et récurrent

Partie Interface Graphique :

Pour réaliser cette partie, nous avons utilisé Qt-Designer

Grâce à Qt-Designer et PyQt, l'implémentation d'une interface deviens assez facile vu que python peut génerer automatiquement le code de l'interface crée avec Qt-Designer, il suffit juste alors d'ajouter le code des fonctions et tout sera fonctionnel

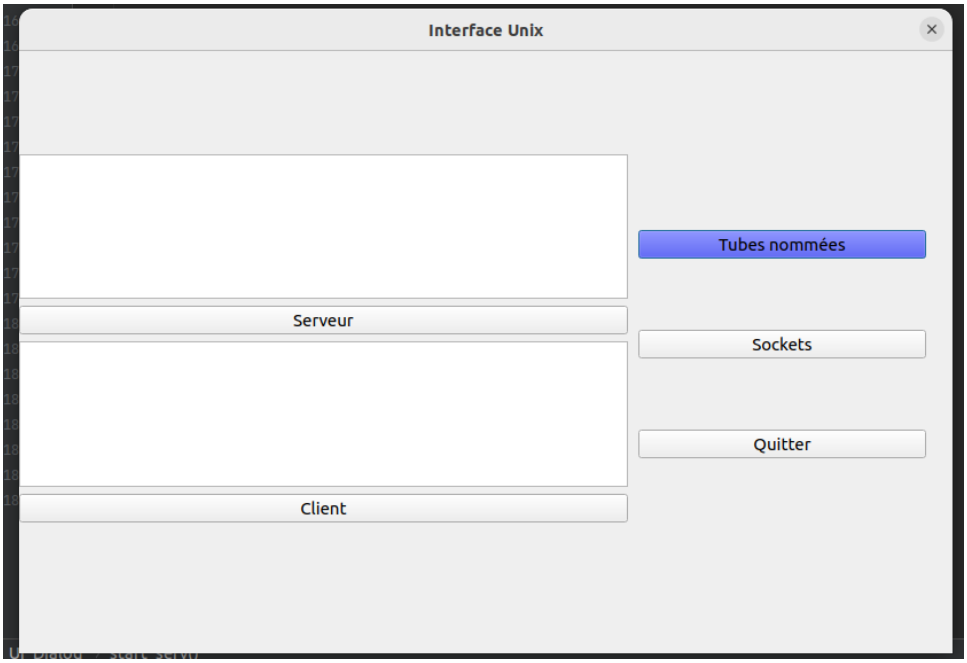
Exemple de code généré par Qt-Designer :

Ci-dessous une partie du code de notre interface contenu dans le fichier python "mainmenu.py" (que vous allez voir après) crée automatiquement grâce à PyQt

```
class Ui_Dialog(object):
    def __init__(self):
        self.choice = 1
        self.server = None
        self.client = None
    def setupUi(self, Dialog):
        Dialog.setObjectName("Interface Unix")
        Dialog.resize(814, 525)
        self.verticalLayoutWidget = QtWidgets.QWidget(Dialog)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(540, 100, 251, 311))
        self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.pushButton_2 = QtWidgets.QPushButton(self.verticalLayoutWidget)
        self.pushButton_2.setObjectName("pushButton_2")
        self.verticalLayout.addWidget(self.pushButton_2)
        self.pushButton = QtWidgets.QPushButton(self.verticalLayoutWidget)
        self.pushButton.setObjectName("pushButton")
        self.verticalLayout.addWidget(self.pushButton)
        self.pushButton_3 = QtWidgets.QPushButton(self.verticalLayoutWidget)
        self.pushButton_3.setObjectName("pushButton_3")
```

Interface finale :

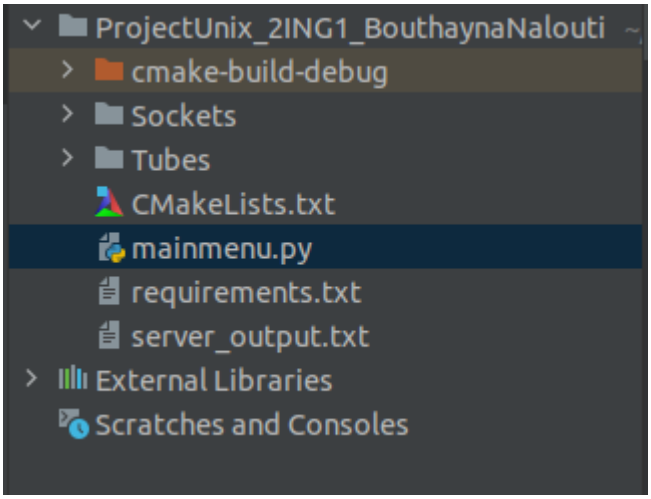
Ci-dessous l'interface finale crée grâce au code et Qt-Designer



Exécution et Partie principale du projet :

Architecture :

Nous avons un total de quatres fichiers enregistrés sous le repertoire “Sockets”. Il y’a un fichier Header, deux fichiers Source et un fichier Makefile qui sont comme suit :Nous avons donc nos deux répertoires des parties précedentes, “Sockets” et “Tubes”, deux fichier textes et finalement un fichier Python



Sockets/Tubes : Les deux repertoires contenant respectivement la partie Socket du projet et la partie Tubes

mainmenu.py : C'est le fichier principal du projet, l'interface est ouverte à partir de ce fichier.

requirements.txt : Fichier contenant les modules necessaires à installer afin de faciliter l'usage de notre programme à tout les utilisateurs

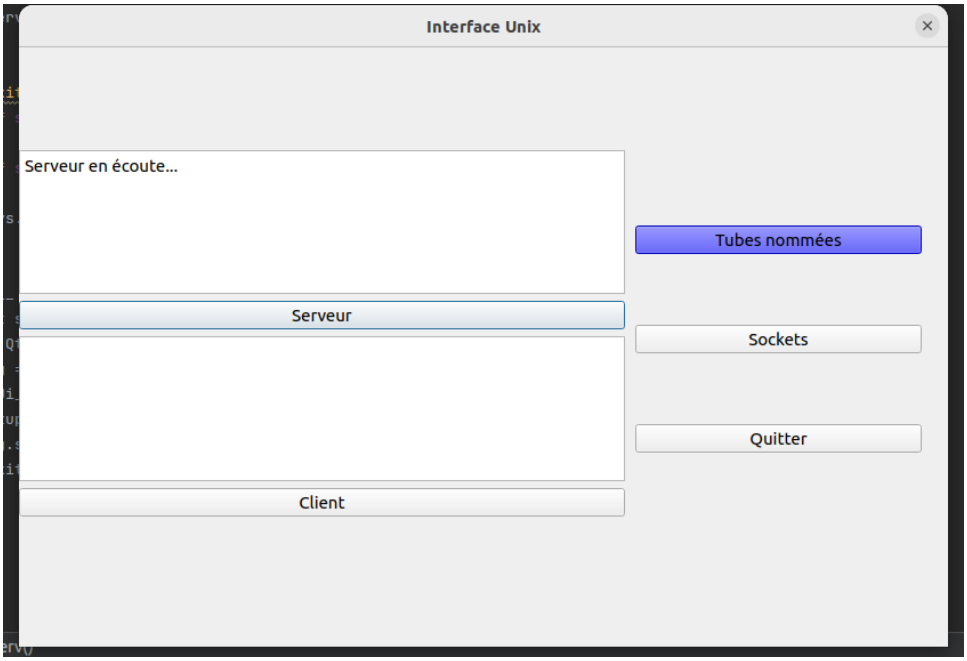
server_output.txt : Fichier contenant l'output du serveur en cours d'utilisation, nécessaire pour l'affichage à temps réel du contenu du serveur dans l'interface

Exécution :

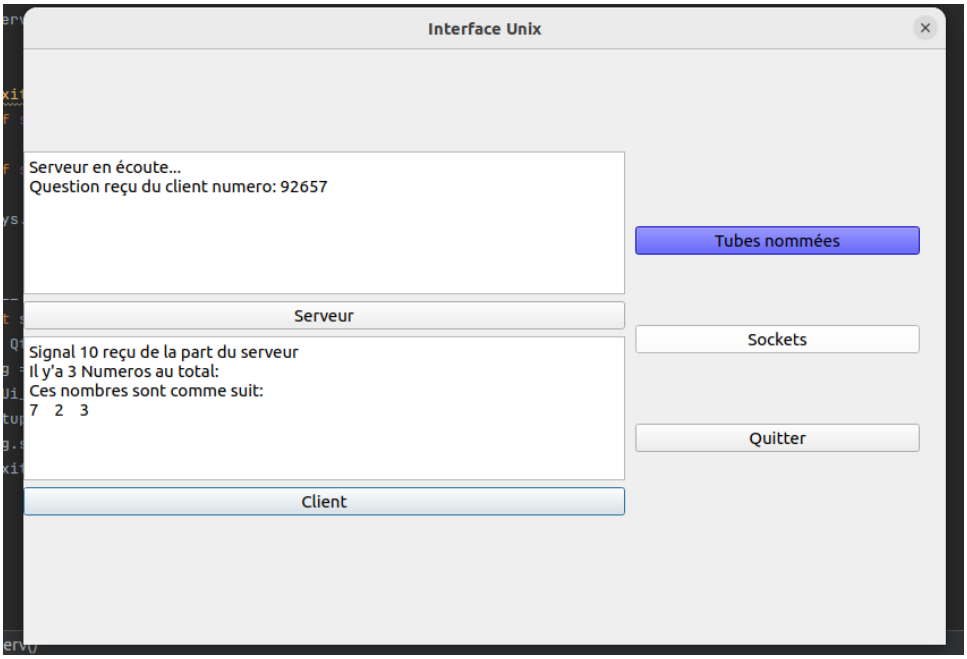
Afin d'exécuter notre programme, il suffit tout simplement d'executer le fichier mainmenu.py et de choisir le traitement voulu dans l'interface. Ci-dessous deux exemples pour les deux parties du projet

Test d'exécution de la partie Tubes nommées :

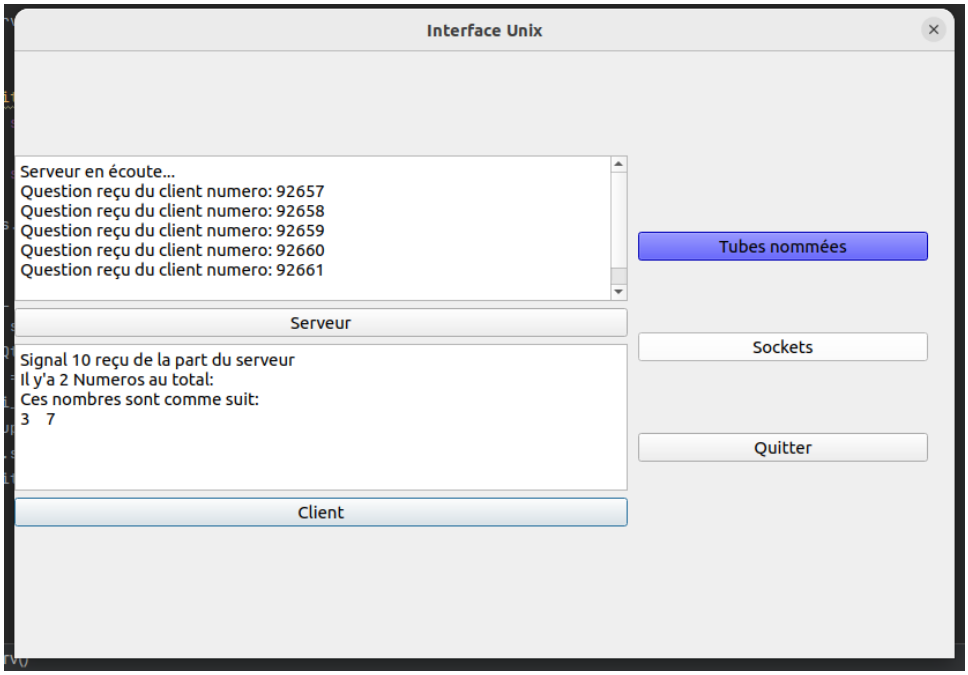
Initialisation du serveur :



Connexion du client et traitement :

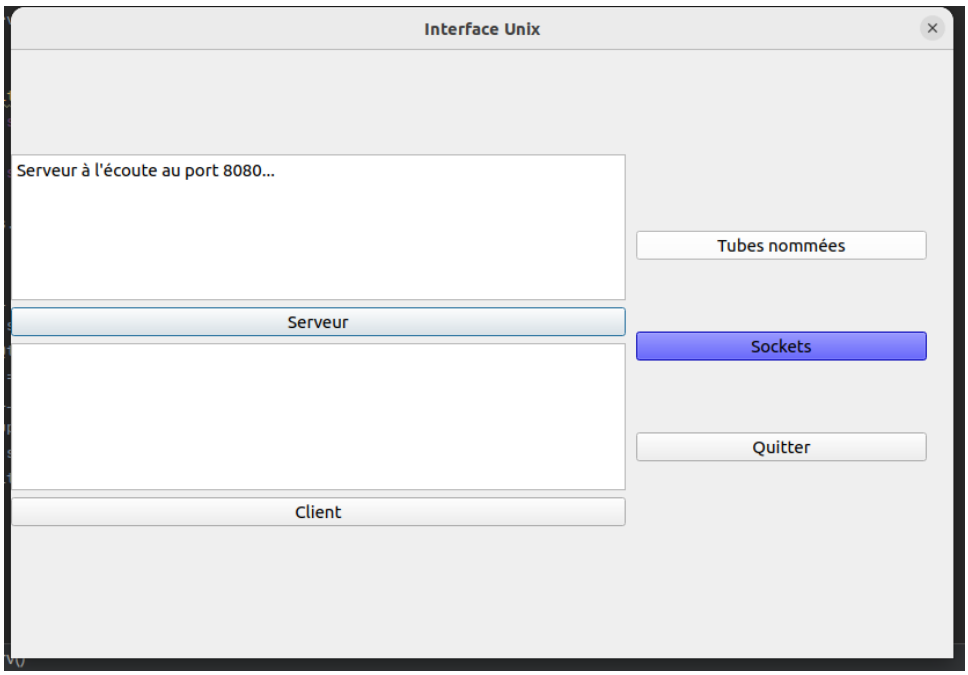


Connexion d’autres clients et traitement :

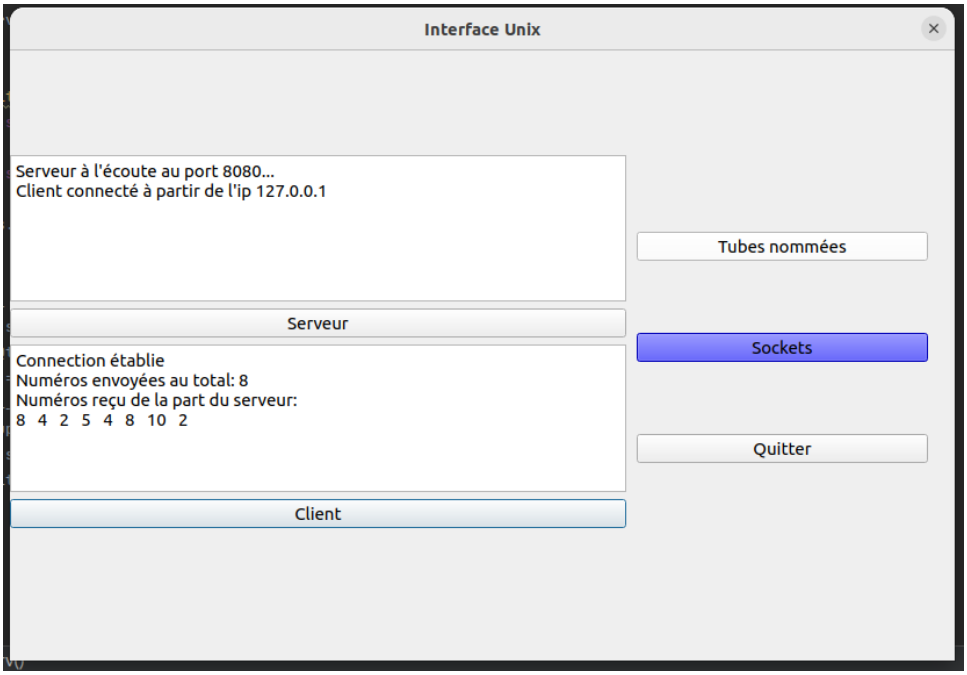


Test d'exécution de la partie Tubes nommées :

Initialisation du serveur :



Connexion du client et traitement :



Connexion d'autres clients et traitement :