MINISTERE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ TUNIS EL MANAR
INSTITUT SUPÉRIEUR D'INFORMATIQUE

Travail

Rapport du mini projet

Projet

Simulation des communication inter-process

Étudiants

Ammar Mohamed Ayoub

BEN NSIB Nassim

Classe

2 ING 02

Enseignant

ZAGROUBA Ezzeddine

Année universitaire

2022-2023

Plan

1.Contexte
2.Technologies
3.Structure du projet
4. Jeu d'essai
5.Implementation

1.Contexte :

La communication inter-processus (inter-process communication, IPC, en anglais) regroupe un ensemble de mécanismes permettant à des processus concurrents de communiquer. Ces mécanismes peuvent être classés en trois catégories :

- Les mécanismes permettant l'échange de données entre les processus ;
- Les mécanismes permettant la synchronisation entre les processus (notamment pour gérer le principe de section critique) ;
- Les mécanismes permettant l'échange de données et la synchronisation entre les processus.

2.Technologies :

Dans cette partie, nous présenterons les différents outils et langages utilisés dans ce projet.

- Langage C:
- **Le langage C** reste un des langages les plus utilisés actuellement. Cela est dû au fait que le langage C est un langage comportant des instructions et des structures de haut niveau tout en générant un code très rapide grâce à un compilateur très performant.

- Langage Python 3
- Python 3 est un langage de programmation de haut niveau à usage général. Sa philosophie de conception met l'accent sur la lisibilité du code avec l'utilisation d'une indentation importante. Python est typé dynamiquement et recueille les déchets. Il prend en charge plusieurs paradigmes de programmation, notamment la programmation structurée, orientée objet et fonctionnelle.

- Oracle Virtual Machine (VM):
- Une machine virtuelle est un environnement virtualisé qui fonctionne sur une machine physique. Elle permet d'émuler un OS sans l'installer physiquement sur l'ordinateur.

- Tkinter Python
- Tkinter est une liaison Python à la boîte à outils Tk GUI. C'est l'interface Python standard de la boîte à outils Tk GUI, et c'est l'interface graphique standard de facto de Python. Tkinter est inclus dans les installations standard de Python pour Linux, Microsoft Windows et macOS. Le nom Tkinter vient de Tk interface.

- Ubuntu :

  C'est un système d'exploitation système d'exploitation libre et open-source.

3.Structure du projet :

a) Capture

b) Description

Le projet contient 2 répertoires (**coeur-project** et **build**) et 2 fichiers (**makefile** et **graphique_application.py**) :

- ***makefile*** : pour automatiser la compilation séparée des modules et la génération d'un exécutable.

- ***graphique_application.py*** : Fichier de l'interface principale.

- **coeur-projet** : Contient le cœur du notre projet ; 2 repertoires (**communication_inter-processus** et **graphique**)

- **graphique :** Contient les interfaces des clients et du serveur et un répertoire ressources inclut les images utilisées.

- **communication_inter-processus :** Les 2 parties 1 (communication via les tubes nommées) et 2 (communication via les sockets avec les deux protocoles TCP & UDP).

- **lib :** Contient les fichiers des constantes, les structures de données et les opérations/méthodes utilisées dans les différents parties du projet.

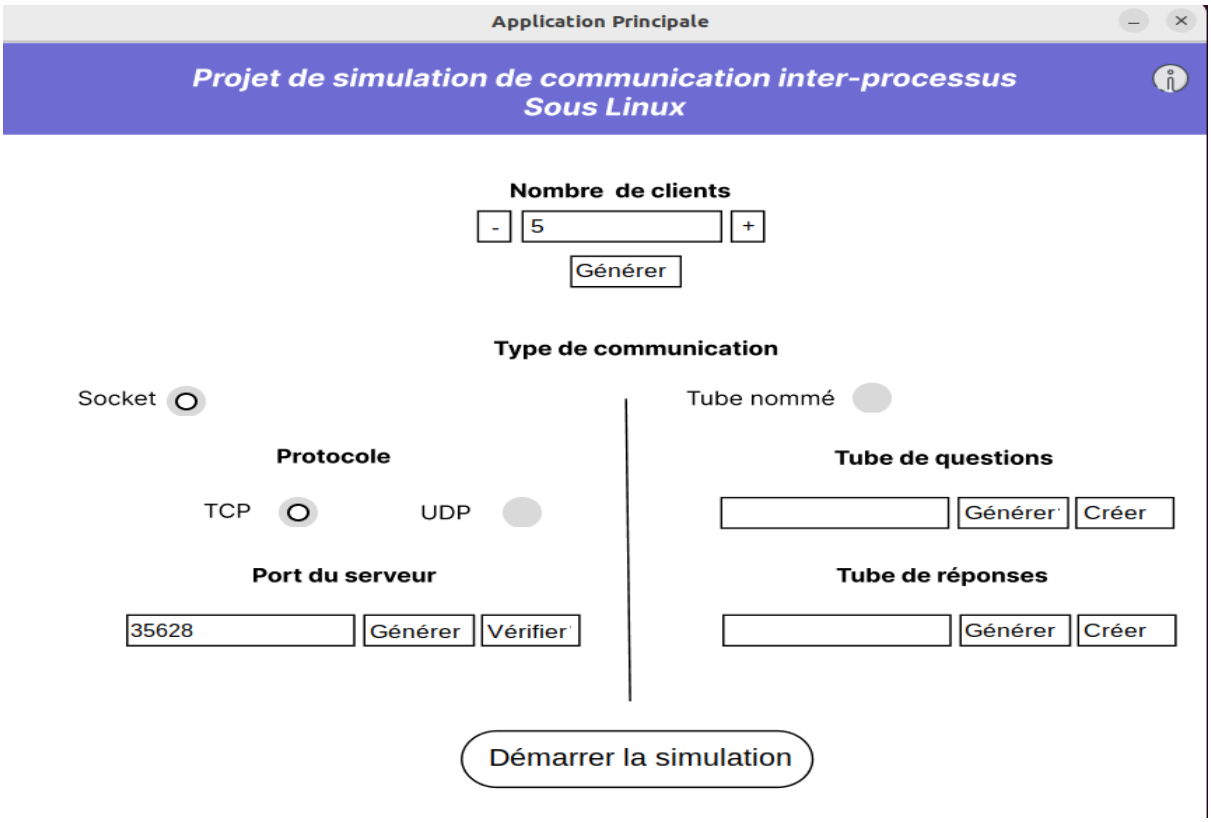
- **Socket / Tube nommée :** Contient la logique de communication inter-processus.

4. Jeu d'essai :

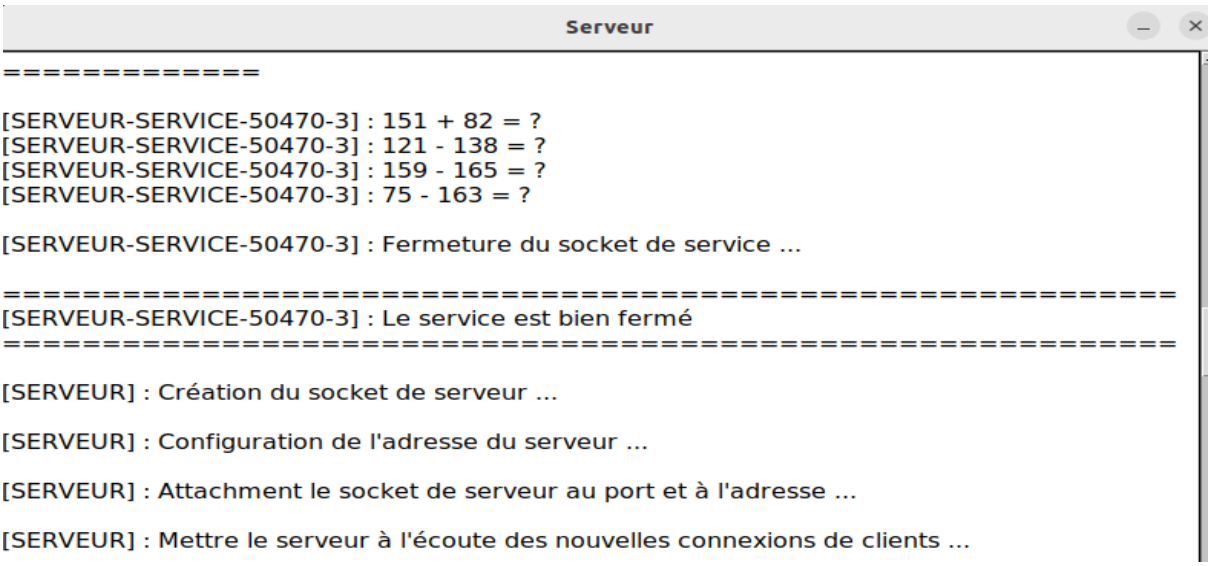
1) Makefile

```
super@iphone:/media/super/Data/labs/projects/linux$ sudo make
[sudo] password for super:
------------------------------------------------------------------------------------------------------------
[DEBUT] : Suppression de la trace de l'ancienne construction
------------------------------------------------------------------------------------------------------------

rmdir -p build/temp

------------------------------------------------------------------------------------------------------------
[FIN] : Suppression de la trace de l'ancienne construction
------------------------------------------------------------------------------------------------------------



------------------------------------------------------------------------------------------------------------
[DENUT] : Compilation de fichiers
------------------------------------------------------------------------------------------------------------

gcc -o ./build/client-TCP ./cœur_projet/communication_inter-processus/socket/protocole-TCP/client-TCP.c
gcc -o ./build/client-UDP ./cœur_projet/communication_inter-processus/socket/protocole-UDP/client-UDP.c
gcc -o ./build/client-FIFO ./cœur_projet/communication_inter-processus/tube_nomme/client-FIFO.c
gcc -o ./build/serveur-TCP ./cœur_projet/communication_inter-processus/socket/protocole-TCP/serveur-TCP.c
gcc -o ./build/serveur-UDP ./cœur_projet/communication_inter-processus/socket/protocole-UDP/serveur-UDP.c
gcc -o ./build/serveur-FIFO ./cœur_projet/communication_inter-processus/tube_nomme/serveur-FIFO.c

------------------------------------------------------------------------------------------------------------
[FIN] : Compilation de fichiers
------------------------------------------------------------------------------------------------------------


============================================================================================================
Projet est prêt
============================================================================================================
```

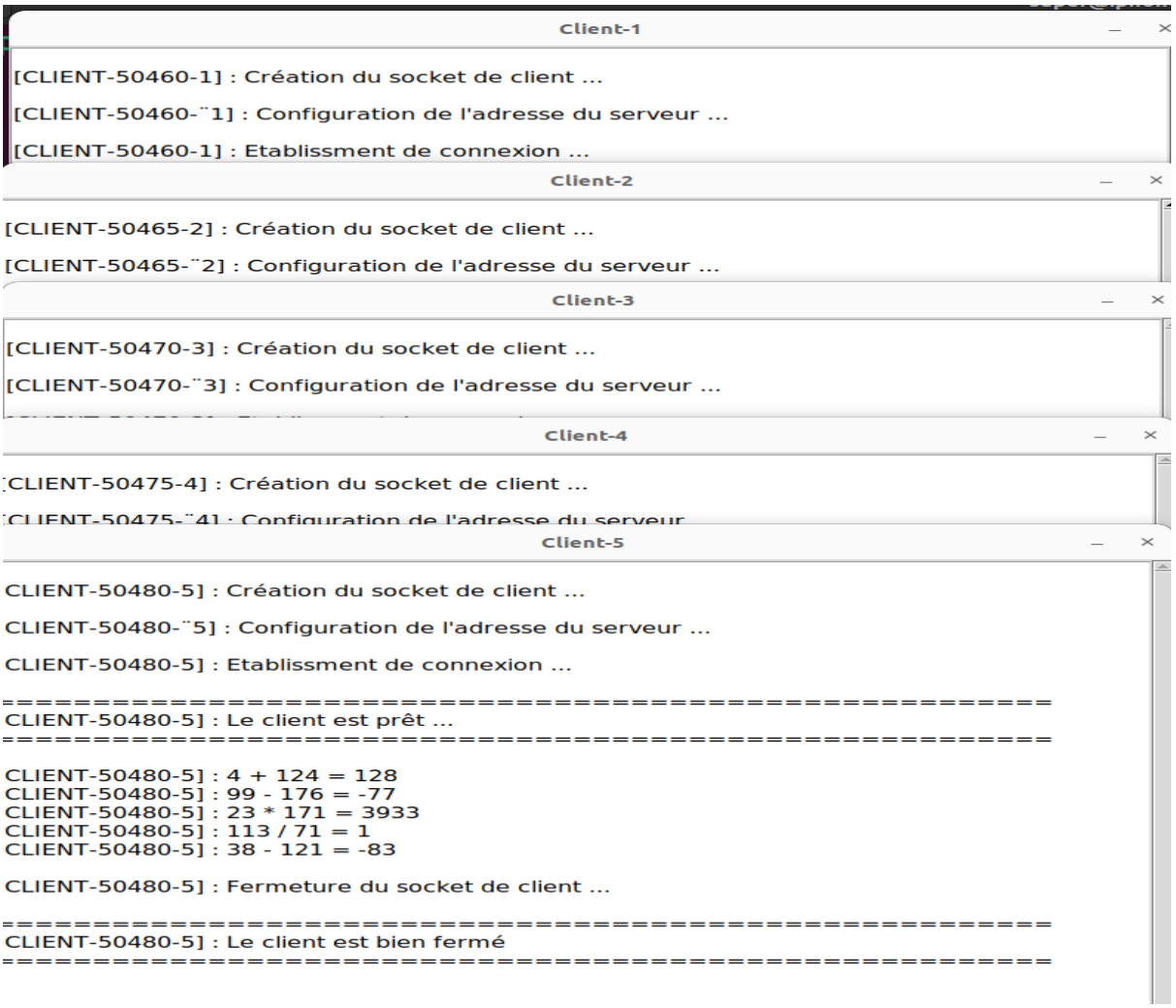
2) Application principale



3) Serveur



4) Clients



5. Implementation:

a) Tube nommée

a.1) Serveur

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/stat.h>
#include <signal.h>
#include <errno.h>
#include <fcntl.h>
#include "./../lib/types.h"
#include "./../lib/operations.h"
#include "./../lib/constantes.h"

int main(int argc, char const *argv[])
{

   int descripteur_tube_question, descripteur_tube_reponse,
resultat_operation, nombre_client = atoi(argv[1]);
   question question;
   reponse reponse;
   info info_client, info_serveur;

   // Installation des handlers
   printf("[SERVEUR] : Installation des handlers...\n\n");
   signal(SIGUSR1, handler);

   // Ouverture de tube (question & reponse)
   printf("[SERVEUR] : Ouverture de tube (question & réponse)
...\n\n");
   descripteur_tube_question = resultat_operation = open(argv[2],
O_RDONLY);
   traiter_erreur(resultat_operation, "OPEN", "Erreur lors de
l'ouverture du tube de questions");
   descripteur_tube_reponse = resultat_operation = open(argv[3],
O_WRONLY);
   traiter_erreur(resultat_operation, "OPEN", "Erreur lors de
l'ouverture du tube de réponses");
```

```
   printf("=========================================================\n");
   printf("[SERVEUR] : Le serveur est prêt\n");

printf("=========================================================\n\n");

   while (nombre_client)
   {
       // Lecture des informations du client
       resultat_operation = read(descripteur_tube_question,
&info_client, sizeof(info));
       traiter_erreur(resultat_operation, "READ", "Erreur lors de la
lecture des informations du client");

       if (resultat_operation != 0)
       {

           // Envoie des informations du serveur
           info_serveur.pid_source = getpid();
           resultat_operation = write(descripteur_tube_reponse,
&info_serveur, sizeof(info));
           traiter_erreur(resultat_operation, "WRITE", "Erreur lors de
l'envoie des informations du serveur");

           // Envoie signal au client
           kill(info_client.pid_source, SIGUSR1);

           while (info_client.nombre_question--)
           {

               // Attente du signal de client
               pause();

               //  Lecture du question
               resultat_operation = read(descripteur_tube_question,
&question, sizeof(question));
               traiter_erreur(resultat_operation, "READ", "Erreur lors
de la lecture du question");
               printf("[SERVEUR-CLIENT-%d-%d] : %d %s %d = ?\n",
question.pid_client, question.numero_client, question.operande_1,
operateurs[question.operation], question.operande_2);
               fflush(stdout);

               // Génération de réponse
```

```
                reponse = generer_reponse(question, 0);

                // Envoie de réponse
                resultat_operation = write(descripteur_tube_reponse,
&reponse, sizeof(reponse));
                traiter_erreur(resultat_operation, "WRITE", "Erreur lors
de l'envoie de la réponse");

                // Envoie signal au client
                kill(info_client.pid_source, SIGUSR1);
            }
            nombre_client--;

printf("=======================================================\n\n");
        }
    }

    // Fermeture des tubes
    printf("[SERVEUR] : Fermeture des tubes ...\n\n");
    resultat_operation = close(descripteur_tube_question);
    traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du tube de questions");
    resultat_operation = close(descripteur_tube_reponse);
    traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du tube de réponses");
    printf("=======================================================\n");
    printf("[SERVEUR] : Le serveur est bien fermé \n");
    printf("=======================================================\n");

    return 0;
}
```

a.2) Client

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/stat.h>
#include <signal.h>
```

```c
#include <errno.h>
#include <fcntl.h>
#include "./../lib/types.h"
#include "./../lib/operations.h"
#include "./../lib/constantes.h"

int main(int argc, char const *argv[])
{

   int descripteur_tube_question, descripteur_tube_reponse,
resultat_operation, numero_client = atoi(argv[1]);
   question question;
   reponse reponse;
   info info_client, info_serveur;
   srand(getpid());

   // Installation des handlers
   printf("[CLIENT] : Installation des handlers...\n\n");
   signal(SIGUSR1, handler);

   // Ouverture de tube (question & reponse)
   printf("[CLIENT] : Ouverture de tube (question & réponse) ...\n\n");
   descripteur_tube_question = resultat_operation = open(argv[2],
O_WRONLY);
   traiter_erreur(resultat_operation, "OPEN", "Erreur lors de
l'ouverture du tube de questions");
   descripteur_tube_reponse = resultat_operation = open(argv[3],
O_RDONLY);
   traiter_erreur(resultat_operation, "OPEN", "Erreur lors de
l'ouverture du tube de réponses");

   printf("=======================================================\n");
   printf("[CLIENT] : Le client est prêt\n");

printf("=======================================================\n\n");

   // Envoie des informations du client
   info_client = generer_nombre_question(getpid(), numero_client);
   resultat_operation = write(descripteur_tube_question, &info_client,
sizeof(info_client));
   traiter_erreur(resultat_operation, "WRITE", "Erreur lors de l'envoie
des informations du client");
```

```
   // Attente du signal de serveur
   pause();

   // Lecture des informations du serveur
   resultat_operation = read(descripteur_tube_reponse, &info_serveur,
sizeof(info_serveur));
   traiter_erreur(resultat_operation, "READ", "Erreur lors de la
lecture des informations du serveur");

   while (info_client.nombre_question--)
   {

       //  Envoie du question
       question = generer_question(getpid(), numero_client, 0);
       resultat_operation = write(descripteur_tube_question, &question,
sizeof(question));
       traiter_erreur(resultat_operation, "WRITE", "Erreur lors de
l'écriture du question");
       printf("[CLIENT-%d-%d] : %d %s %d = ", question.pid_client,
question.numero_client, question.operande_1,
operateurs[question.operation], question.operande_2);
       fflush(stdout);

       // Envoie signal au serveur
       kill(info_serveur.pid_source, SIGUSR1);

       // Attente du signal de serveur
       pause();

       // Lecture de réponse
       resultat_operation = read(descripteur_tube_reponse, &reponse,
sizeof(reponse));
       traiter_erreur(resultat_operation, "READ", "Erreur lors de la
lecture de la réponse");
       printf("%d\n", reponse.resultat);
       fflush(stdout);

       // Envoie signal au serveur
       kill(info_serveur.pid_source, SIGUSR1);
   }

   // Fermeture des tubes
   printf("\n[CLIENT] : Fermeture des tubes ...\n\n");
```

```
   resultat_operation = close(descripteur_tube_question);
   traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du tube de questions");
   resultat_operation = close(descripteur_tube_reponse);
   traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du tube de réponses");
   printf("=========================================================\n");
   printf("[CLIENT] : Le client est bien fermé \n");
   printf("=========================================================\n");

   return 0;
}
```

b) Socket - TCP

b.1) Serveur

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include "./../../lib/types.h"
#include "./../../lib/operations.h"
#include "./../../lib/constantes.h"


void lancer_service_client(int descripteur_socket_client)
{
   int resultat_operation;
   reponse reponse;
   question question;
   info info;

   // Lecture des informations du client
   resultat_operation = read(descripteur_socket_client, &info,
sizeof(info));
   traiter_erreur(resultat_operation, "READ", "Erreur lors de la
lecture des informations du client");
```

```
printf("================================================================
=========\n");
   printf("[SERVEUR-SERVICE-%d-%d] : Service est prêt pour le client
...\n", info.pid_source, info.numero_client);

printf("================================================================
=========\n\n");

   for (int numero = 1; numero <= info.nombre_question; numero++)
   {
       // Lecture du question
       resultat_operation = read(descripteur_socket_client, &question,
sizeof(question));
       traiter_erreur(resultat_operation, "READ", "Erreur lors de la
lecture du question");
       printf("[SERVEUR-SERVICE-%d-%d] : %d %s %d = ?\n",
info.pid_source, info.numero_client, question.operande_1,
operateurs[question.operation], question.operande_2);

       // Génération de réponse
       reponse = generer_reponse(question, 0);

       // Envoie de réponse
       resultat_operation = write(descripteur_socket_client, &reponse,
sizeof(reponse));
       traiter_erreur(resultat_operation, "READ", "Erreur lors de
l'envoie de la réponse");

       fflush(stdout);
   }

   // Fermeture du socket de client
   printf("\n[SERVEUR-SERVICE-%d-%d] : Fermeture du socket de service
...\n\n", info.pid_source, info.numero_client);
   resultat_operation = close(descripteur_socket_client);
   traiter_erreur(descripteur_socket_client, "CLOSE", "Erreur lors de
la fermeture du socket de service");

printf("============================================================\n")
;
   printf("[SERVEUR-SERVICE-%d-%d] : Le service est bien fermé\n",
info.pid_source, info.numero_client);
```

```
printf("============================================================\n\n
");
   exit(0);
   fflush(stdout);
}

int main(int argc, char const *argv[])
{

   int descripteur_socket_serveur, descripteur_socket_client,
resultat_operation, nombre_client = atoi(argv[1]),port_serveur =
atoi(argv[2]);
   struct sockaddr_in adresse_socket_serveur, adresse_socket_client;
   socklen_t taille_adresse_client = sizeof(adresse_socket_client);

   // Création du socket de serveur
   printf("[SERVEUR] : Création du socket de serveur ...\n\n");
   descripteur_socket_serveur = resultat_operation = socket(AF_INET,
SOCK_STREAM, 0);
   traiter_erreur(resultat_operation, "SOCKET", "Erreur lors de
création du socket de serveur");

   // Configuration d'adresse de serveur
   printf("[SERVEUR] : Configuration de l'adresse du serveur ...\n\n");
   bzero((char *)&adresse_socket_serveur,
sizeof(adresse_socket_serveur));
   adresse_socket_serveur.sin_family = AF_INET;
   adresse_socket_serveur.sin_addr.s_addr = INADDR_ANY;
   adresse_socket_serveur.sin_port = htons(port_serveur =
atoi(argv[2]));

   // Attachment le socket de serveur au port et à l'adresse
   printf("[SERVEUR] : Attachment le socket de serveur au port et à
l'adresse ...\n\n");
   resultat_operation = bind(descripteur_socket_serveur, (struct
sockaddr *)&adresse_socket_serveur, sizeof(adresse_socket_serveur));
   traiter_erreur(resultat_operation, "BIND", "Erreur lors d'attachment
le socket de serveur au port et à l'adresse");

   // Mettre le serveur à l'écoute des nouvelles connexions de clients
   printf("[SERVEUR] : Mettre le serveur à l'écoute des nouvelles
connexions de clients ...\n\n");
```

```
   resultat_operation = listen(descripteur_socket_serveur,
NOMBRE_MAX_CONNEXION_CLIENT);
   traiter_erreur(resultat_operation, "LISTEN", "Erreur lors de mettre
le serveur à l'écoute des nouvelles connexions de clients");

   printf("=========================================================\n");
   printf("[SERVEUR] : Le serveur est prêt\n");

printf("=========================================================\n\n");

   while (nombre_client--)
   {
       // Attente de nouvelle connexion
       printf("\n[SERVEUR] : Attente de nouvelles connexions ...\n\n");
       descripteur_socket_client = resultat_operation =
accept(descripteur_socket_serveur, (struct sockaddr
*)&adresse_socket_client, &taille_adresse_client);
       traiter_erreur(resultat_operation, "ACCEPT", "Erreur lors de
l'acceptation de nouvelle connexion");

       // Création de nouveau processus fils, pour s'occuper de nouveau
client
       printf("[SERVEUR] : Création de nouveau processus fils, pour
s'occuper de nouveau client ...\n\n");
       resultat_operation = fork();
       traiter_erreur(resultat_operation, "FORK", "Erreur lors de la
création nouveau processus fils");

       // Lancement de logique de service
       if (!resultat_operation)
       {
           lancer_service_client(descripteur_socket_client);
       }
   }

   // Fermeture du socket de serveur
   printf("[SERVEUR] : Fermeture du socket de serveur ...\n\n");
   resultat_operation = shutdown(descripteur_socket_serveur,
SHUT_RDWR);
   traiter_erreur(resultat_operation, "SHUTDOWN", "Erreur lors de la
fermeture du socket de client");
   resultat_operation = close(descripteur_socket_serveur);
```

```
    traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du socket de client");
    printf("============================================================\n");
    printf("[SERVEUR] : Le serveur est bien fermé \n");
    printf("============================================================\n");

    return 0;
}
```

b.2) Client

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include "./../../lib/types.h"
#include "./../../lib/operations.h"
#include "./../../lib/constantes.h"


int main(int argc, char const *argv[])
{
    int descripteur_socket_client, resultat_operation, numero_client =
atoi(argv[1]),port_serveur = atoi(argv[2]);
    struct sockaddr_in adresse_socket_serveur;
    info info;
    question question;
    reponse reponse;
    srand(getpid());

    // Création du socket de client
    printf("\n[CLIENT-%d-%d] : Création du socket de client ...\n\n",
getpid(), numero_client);
    descripteur_socket_client = resultat_operation = socket(AF_INET,
SOCK_STREAM, 0);
    traiter_erreur(resultat_operation, "SOCKET", "Erreur lors de
création du socket de client");

    // Configuration d'adresse de serveur
    printf("[CLIENT-%d-¨%d] : Configuration de l'adresse du serveur
...\n\n", getpid(), numero_client);
```

```
   bzero((char *)&adresse_socket_serveur,
sizeof(adresse_socket_serveur));
   adresse_socket_serveur.sin_family = AF_INET;
   adresse_socket_serveur.sin_port = htons(port_serveur);

   // Conversion de l'adresse IPV4 en forme binaire
   resultat_operation = inet_pton(AF_INET, "127.0.0.1", (struct
sockaddr *)&adresse_socket_serveur.sin_addr);
   traiter_erreur(resultat_operation, "INET_PTON", "Erreur lors de la
conversion de l'adresse IPV4 en forme binaire");

   // Etablissment de connexion
   printf("[CLIENT-%d-%d] : Etablissment de connexion ...\n\n",
getpid(), numero_client);
   resultat_operation = connect(descripteur_socket_client, (struct
sockaddr *)&adresse_socket_serveur, sizeof(adresse_socket_serveur));
   traiter_erreur(resultat_operation, "CONNECT", "Erreur lors
d'etablissment de connexion");

   printf("=========================================================\n");
   printf("[CLIENT-%d-%d] : Le client est prêt ...\n", getpid(),
numero_client);

printf("=========================================================\n\n");

   info = generer_nombre_question(getpid(), numero_client);
   resultat_operation = write(descripteur_socket_client, &info,
sizeof(info));
   traiter_erreur(resultat_operation, "WRITE", "Erreur lors de l'envoie
des informations du client");

   for (int numero = 1; numero <= info.nombre_question; numero++)
   {
       // Génération du question
       question = generer_question(getpid(), 0, 0);
       printf("[CLIENT-%d-%d] : %d %s %d = ", getpid(), numero_client,
question.operande_1, operateurs[question.operation],
question.operande_2);

       // Envoie du question
       resultat_operation = write(descripteur_socket_client, &question,
sizeof(question));
```

```
        traiter_erreur(resultat_operation, "WRITE", "Erreur lors de
l'envoie du question");

        // Lecture de la réponse
        resultat_operation = read(descripteur_socket_client, &reponse,
sizeof(reponse));
        traiter_erreur(resultat_operation, "READ", "Erreur lors de la
lecture de la réponse");
        printf("%d\n", reponse.resultat);

        fflush(stdout);
    }

    // Fermeture du socket de client
    printf("\n[CLIENT-%d-%d] : Fermeture du socket de client ...\n\n",
getpid(), numero_client);
    resultat_operation = close(descripteur_socket_client);
    traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du socket de client");
    printf("=========================================================\n");
    printf("[CLIENT-%d-%d] : Le client est bien fermé\n", getpid(),
numero_client);

printf("=========================================================\n\n");

    return 0;
}
```

c) Socket - UDP

c.1) Serveur

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include "./../../lib/types.h"
#include "./../../lib/operations.h"
```

```c
#include "./../../lib/constantes.h"

int main(int argc, char const *argv[])
{

   int descripteur_socket_serveur, descripteur_socket_client,
resultat_operation, nombre_client = atoi(argv[1]),port_serveur =
atoi(argv[2]);
   struct sockaddr_in adresse_socket_serveur, adresse_socket_client;
   int taille_adresse_client = sizeof(adresse_socket_client);
   question question;
   reponse reponse;

   // Création du socket de serveur
   printf("[SERVEUR] : Création du socket de serveur ...\n\n");
   descripteur_socket_serveur = resultat_operation = socket(AF_INET,
SOCK_DGRAM, 0);
   traiter_erreur(resultat_operation, "SOCKET", "Erreur lors de
création du socket de serveur");

   // Configuration d'adresse de serveur
   printf("[SERVEUR] : Configuration de l'adresse du serveur ...\n\n");
   bzero((char *)&adresse_socket_serveur,
sizeof(adresse_socket_serveur));
   bzero((char *)&adresse_socket_client,
sizeof(adresse_socket_client));
   adresse_socket_serveur.sin_family = AF_INET;
   adresse_socket_serveur.sin_addr.s_addr = INADDR_ANY;
   adresse_socket_serveur.sin_port = htons(port_serveur);

   // Attachment le socket de serveur au port et à l'adresse
   printf("[SERVEUR] : Attachment le socket de serveur au port et à
l'adresse ...\n\n");
   resultat_operation = bind(descripteur_socket_serveur, (struct
sockaddr *)&adresse_socket_serveur, sizeof(adresse_socket_serveur));
   traiter_erreur(resultat_operation, "BIND", "Erreur lors d'attachment
le socket de serveur au port et à l'adresse");

   printf("=========================================================\n");
   printf("[SERVEUR] : Le serveur est prêt\n");

printf("=========================================================\n\n");
```

```
    while (nombre_client)
    {
        // Lecture du question
        resultat_operation = recvfrom(descripteur_socket_serveur,
&question, sizeof(question), MSG_WAITALL, (struct sockaddr
*)&adresse_socket_client, &taille_adresse_client);
        traiter_erreur(resultat_operation, "RECVFROM", "Erreur lors de
la lecture du question");
        printf("[SERVEUR-CLIENT-%d-%d] : %d %s %d = ?\n",
question.pid_client, question.numero_client, question.operande_1,
operateurs[question.operation], question.operande_2);
        if (question.est_dernier_question)
        {
            nombre_client--;
        }

        // Génération de réponse
        reponse = generer_reponse(question, 0);

        // Envoie de réponse
        resultat_operation = sendto(descripteur_socket_serveur,
&reponse, sizeof(reponse), MSG_CONFIRM, (const struct sockaddr
*)&adresse_socket_client, taille_adresse_client);
        traiter_erreur(resultat_operation, "SENDTO", "Erreur lors de
l'envoie de la réponse");

        fflush(stdout);
    }

    // Fermeture du socket de serveur
    printf("\n[SERVEUR] : Fermeture du socket de serveur ...\n\n");
    resultat_operation = close(descripteur_socket_serveur);
    traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du socket de client");
    printf("=========================================================\n");
    printf("[SERVEUR] : Le serveur est bien fermé \n");
    printf("=========================================================\n");

    return 0;
}
```

c.2) Client

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include "./../../lib/types.h"
#include "./../../lib/operations.h"
#include "./../../lib/constantes.h"


int main(int argc, char const *argv[])
{

   int descripteur_socket_client, resultat_operation, numero_client =
atoi(argv[1]),port_serveur = atoi(argv[2]);
   struct sockaddr_in adresse_socket_serveur;
   int taille_adresse_serveur = sizeof(adresse_socket_serveur);
   info info;
   question question;
   reponse reponse;
   srand(getpid());

   // Création du socket de client
   printf("\n[CLIENT-%d-%d] : Création du socket de client ...\n\n",
getpid(), numero_client);
   descripteur_socket_client = resultat_operation = socket(AF_INET,
SOCK_DGRAM, 0);
   traiter_erreur(resultat_operation, "SOCKET", "Erreur lors de
création du socket de client");

   // Configuration d'adresse de serveur
   printf("[CLIENT-%d-¨%d] : Configuration de l'adresse du serveur
...\n\n", getpid(), numero_client);
   bzero((char *)&adresse_socket_serveur,
sizeof(adresse_socket_serveur));
   adresse_socket_serveur.sin_family = AF_INET;
   adresse_socket_serveur.sin_port = htons(port_serveur);

   printf("=========================================================\n");
```

```
   printf("[CLIENT-%d-%d] : Le client est prêt ...\n", getpid(),
numero_client);

printf("=========================================================\n\n");

   info = generer_nombre_question(getpid(), 0);

   for (int numero = 1; numero <= info.nombre_question; numero++)
   {
       // Génération du question
       question = generer_question(getpid(), numero_client, numero ==
info.nombre_question);
       printf("[CLIENT-%d-%d] : %d %s %d = ", getpid(), numero_client,
question.operande_1, operateurs[question.operation],
question.operande_2);

       // Envoie du question
       resultat_operation = sendto(descripteur_socket_client,
&question, sizeof(question), MSG_CONFIRM, (const struct sockaddr
*)&adresse_socket_serveur, taille_adresse_serveur);
       traiter_erreur(resultat_operation, "SENDTO", "Erreur lors de
l'envoie du question");

       // Lecture de la réponse
       resultat_operation = recvfrom(descripteur_socket_client,
&reponse, sizeof(reponse), MSG_WAITALL, (struct sockaddr
*)&adresse_socket_serveur, &taille_adresse_serveur);
       traiter_erreur(resultat_operation, "RECVFROM", "Erreur lors de
la lecture de la réponse");
       printf("%d\n", reponse.resultat);

       fflush(stdout);
   }

   // Fermeture du socket de client
   printf("\n[CLIENT-%d-%d] : Fermeture du socket de client ...\n\n",
getpid(), numero_client);
   resultat_operation = close(descripteur_socket_client);
   traiter_erreur(resultat_operation, "CLOSE", "Erreur lors de la
fermeture du socket de client");
   printf("=========================================================\n");
   printf("[CLIENT-%d-%d] : Le client est bien fermé\n", getpid(),
numero_client);
```

```
printf("============================================================\n\n");
   return 0;
}
```

d) Libraries

d.1) constantes.h

```
#ifndef CONSTANTES_DEFINE_HEADER
#define CONSTANTES_DEFINE_HEADER


#define NOMBRE_MIN_QUESTION 2
#define NOMBRE_MAX_QUESTION 5
#define NOMBRE_OPERATIONS 4
#define VALEUR_MIN_OPERANDE 1
#define VALEUR_MAX_OPERANDE 200

#define PORT_SERVEUR 12345
#define NOMBRE_MAX_CONNEXION_CLIENT 100


#endif
```

d.2) operations_math.h

```
#ifndef OPERATIONS_MATH_DEFINE_HEADER
#define OPERATIONS_MATH_DEFINE_HEADER

int addition(int x, int y)
{
   return x + y;
}

int soustraction(int x, int y)
{
   return x - y;
}

int division(int x, int y)
```

```
{
   return x / y;
}

int multiplication(int x, int y)
{
   return x * y;
}

#endif
```

d.3) operations.h

```
#ifndef OPERATIONS_DEFINE_HEADER
#define OPERATIONS_DEFINE_HEADER

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include "types.h"
#include "constantes.h"

reponse generer_reponse(question question, int pid_serveur)
{
   reponse reponse;

   reponse.resultat =
operations[question.operation](question.operande_1,
question.operande_2);
   reponse.pid_serveur = pid_serveur;

   return reponse;
}

info generer_nombre_question(int pid_client, int numero_client)
{
   info info;

   info.nombre_question = rand() % NOMBRE_MAX_QUESTION;
   info.nombre_question += NOMBRE_MIN_QUESTION;
   info.pid_source = pid_client;
   info.numero_client = numero_client;

   return info;
```

```
}

question generer_question(int pid_client, int numero_client, int
est_dernier_question)
{
   question question;

   question.pid_client = pid_client;
   question.operation = rand() % NOMBRE_OPERATIONS;
   question.operande_1 = rand() % VALEUR_MAX_OPERANDE;
   question.operande_1 += VALEUR_MIN_OPERANDE;
   question.operande_2 = rand() % VALEUR_MAX_OPERANDE;
   question.operande_2 += VALEUR_MIN_OPERANDE;
   question.numero_client = numero_client;
   question.est_dernier_question = est_dernier_question;

   return question;
}

void traiter_erreur(int code, char *operation, char *message_erreur)
{
   if (code < 0)
   {
       printf("\033[91m");
       perror(operation);
       printf("\n%s\n\n", message_erreur);
       exit(EXIT_FAILURE);
   }
}

void handler(int code)
{
}

#endif
```

d.4) type.h

```
#ifndef TYPE_DEFINE_HEADER
#define TYPE_DEFINE_HEADER

#include "operations_math.h"
```

```c
enum operation
{
   ADDITION,
   SOUSSTRACTION,
   DIVISION,
   MULTIPLICATION,
};

typedef struct info
{
   int nombre_question;
   int pid_source;
   int numero_client;
} info;

typedef struct question
{
   enum operation operation;
   int pid_client;
   int numero_client;
   int operande_1;
   int operande_2;
   int est_dernier_question;
} question;

typedef struct reponse
{
   int pid_serveur;
   int resultat;
} reponse;

typedef int (*fonction)(int x, int y);

fonction operations[] = {
   &addition,
   &soustraction,
   &division,
   &multiplication,
};
char *operateurs[] = {"+", "-", "/", "*"};

#endif
```

e) Partie graphique

e.1) graphique_client.py

```
from tkinter import *
from tkinter.ttk import *
from sys import argv

numero = argv[1]

fenetre = Tk()
fenetre.title(f"Client-{numero}")
fenetre.geometry("800x500")
fenetre.resizable(0,0)


v = Scrollbar(fenetre, orient='vertical')
v.pack(side=RIGHT, fill='y')

texte = Text(fenetre, font=("Georgia, 12"), yscrollcommand=v.set)


with open(f"./build/temp/trace_communicarion_client_{numero}.txt", "r")
as file:
   trace_communicarion = "".join(file.readlines())


texte.insert(END, trace_communicarion)

v.config(command=texte.yview)
texte.pack()

fenetre.mainloop()
```

e.2) graphique_serveur.py

```
from tkinter import *
import time
from tkinter.ttk import *

fenetre = Tk()
fenetre.title("Serveur")
fenetre.geometry("800x500")
fenetre.resizable(0,0)
```

```
v = Scrollbar(fenetre, orient='vertical')
v.pack(side=RIGHT, fill='y')

texte = Text(fenetre, font=("Georgia, 12"), yscrollcommand=v.set)


with open(f"./build/temp/trace_communicarion_serveur.txt", "r") as
file:
   trace_communicarion = "".join(file.readlines())


texte.insert(END, trace_communicarion)

v.config(command=texte.yview)
texte.pack()

fenetre.mainloop()
```

f) Makefile

```
chemin_communication = ./cœur_projet/communication_inter-processus
chemin_tcp = ${chemin_communication}/socket/protocole-TCP
chemin_udp = ${chemin_communication}/socket/protocole-UDP
chemin_fifo = ${chemin_communication}/tube_nomme
chemin_construction = ./build

drapeau_debut : supprimer_trace_construction compiler_fichiers
   @echo
"=================================================================
==================================="
   @echo "Projet est prêt"
   @echo
"=================================================================
==================================="
   @echo ""


supprimer_trace_construction :
   @echo
"-----------------------------------------------------------------
---------------------------------"
   @echo "[DEBUT] : Suppression de la trace de l'ancienne construction"
```

```
	@echo
"--------------------------------------------------------------------------
-----------------------------------"
	@echo ""
	@mkdir -p build/temp
	rmdir -p build/temp
	@echo ""
	@echo
"--------------------------------------------------------------------------
-----------------------------------"
	@echo "[FIN] : Suppression de la trace de l'ancienne construction"
	@echo
"--------------------------------------------------------------------------
-----------------------------------"
	@echo "\n"


compiler_fichiers :
	@echo
"--------------------------------------------------------------------------
-----------------------------------"
	@echo "[DENUT] : Compilation de fichiers"
	@echo
"--------------------------------------------------------------------------
-----------------------------------"
	@echo ""
	@mkdir -p build/temp
	gcc -o ${chemin_construction}/client-TCP ${chemin_tcp}/client-TCP.c
	gcc -o ${chemin_construction}/client-UDP ${chemin_udp}/client-UDP.c
	gcc -o ${chemin_construction}/client-FIFO
${chemin_fifo}/client-FIFO.c
	gcc -o ${chemin_construction}/serveur-TCP
${chemin_tcp}/serveur-TCP.c
	gcc -o ${chemin_construction}/serveur-UDP
${chemin_udp}/serveur-UDP.c
	gcc -o ${chemin_construction}/serveur-FIFO
${chemin_fifo}/serveur-FIFO.c
	@echo ""
	@echo
"--------------------------------------------------------------------------
-----------------------------------"
	@echo "[FIN] : Compilation de fichiers"
```

```
	@echo
"-------------------------------------------------------------------------
-----------------------------------"
	@echo "\n"
```

g) Application principale

```
from tkinter import *
from tkinter.messagebox import *
from PIL import ImageTk, Image
from os import system,path
from time import sleep
from random import randint,choice
import string
import socket

chemin_base = "."

def demarrer_application(fenetre : Tk) -> Tk:
	fenetre.mainloop()


def handler_boutton_moins(etiquette_nombre_client : Label) -> None:
    def logique(event) -> None:
        nombre_client = int(etiquette_nombre_client.cget("text"))
        if(nombre_client > 1):
            etiquette_nombre_client.config(text=str(nombre_client-1))

    return logique

def handler_boutton_plus(etiquette_nombre_client : Label) -> None:
    def logique(event) -> None:
        nombre_client = int(etiquette_nombre_client.cget("text"))
        etiquette_nombre_client.config(text=str(nombre_client+1))

    return logique


def handler_choix_socket() -> None:
    def logique(event) -> None:
        global type_communication,type_socket
        type_communication = "SOCKET"
```

```
        type_socket = "TCP"
        fifo_radio_2.place(x=610,y=302)
        fifo_radio_1.place_forget()
        socket_radio_1.place(x=120,y=305)
        socket_radio_2.place_forget()
        socket_tcp_radio_1.place(x=200,y=405)
        socket_tcp_radio_2.place_forget()
        socket_udp_radio_2.place(x=360,y=405)
        socket_udp_radio_1.place_forget()

    return logique


def handler_choix_socket_tcp() -> None:
    def logique(event) -> None:
        global type_communication,type_socket
        type_communication = "SOCKET"
        type_socket = "TCP"
        fifo_radio_2.place(x=610,y=302)
        fifo_radio_1.place_forget()
        socket_radio_1.place(x=120,y=305)
        socket_radio_2.place_forget()
        socket_tcp_radio_1.place(x=200,y=405)
        socket_tcp_radio_2.place_forget()
        socket_udp_radio_2.place(x=360,y=405)
        socket_udp_radio_1.place_forget()

    return logique


def handler_choix_socket_udp() -> None:
    def logique(event) -> None:
        global type_communication,type_socket
        type_communication = "SOCKET"
        type_socket = "UDP"
        fifo_radio_2.place(x=610,y=302)
        fifo_radio_1.place_forget()
        socket_radio_1.place(x=120,y=305)
        socket_radio_2.place_forget()
        socket_tcp_radio_2.place(x=200,y=405)
        socket_tcp_radio_1.place_forget()
        socket_udp_radio_1.place(x=360,y=405)
        socket_udp_radio_2.place_forget()
```

```
    return logique

def handler_choix_fifo() -> None:
    def logique(event) -> None:
        global type_communication,type_socket
        type_communication = "FIFO"
        type_socket = None
        fifo_radio_1.place(x=610,y=302)
        fifo_radio_2.place_forget()
        socket_radio_2.place(x=120,y=305)
        socket_radio_1.place_forget()
        socket_tcp_radio_2.place(x=200,y=405)
        socket_tcp_radio_1.place_forget()
        socket_udp_radio_2.place(x=360,y=405)
        socket_udp_radio_1.place_forget()

    return logique


def handler_generer_nombre_client(etiquette_nombre_client : Label) ->
None:
    def logique(event) -> None:
        etiquette_nombre_client.config(text=str(randint(3,30)))

    return logique


def handler_generer_port_serveur(champ_port_serveur : Entry) -> None:
    def logique(event) -> None:
        champ_port_serveur.delete(0,END)
        champ_port_serveur.insert(0,str(randint(1,65536)))

    return logique


def handler_generer_nom(champ : Entry) -> None:
    def logique(event) -> None:
        lettres = string.ascii_lowercase
        nom = ''.join(choice(lettres) for i in range(10))
        champ.delete(0,END)
        champ.insert(0,nom)
```

```
    return logique


def handler_verifier_port_serveur(champ : Entry) -> None:
    def logique(event) -> None:
        try:
            port = int(champ.get().strip())
            if(port < 1 or port > 65536 ):
                showerror(title="Erreur", message="Numero port invalide
(Le numéro du port doit être entre 1..65536) !!")
                return

            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


            try:
                s.bind(("127.0.0.1", int(champ.get())))
                showinfo(title="Info",message="Ce port est valide et
disponible")
                s.close()
            except socket.error as e:
                showerror(title="Erreur", message="Ce port est déjà
utilisé !!")

        except Exception as e:
            showerror(title="Erreur", message="Numero port invalide !!")

    return logique


def handler_creer_tube_nomme(champ1 : Entry,champ2 : Entry) -> None:
    def logique(event) -> None:
        nom = champ1.get().strip()
        if(len(nom)==0):
            showerror(title="Erreur", message="Champ vide !!")
        elif(champ1.get().strip() == champ2.get().strip()):
            showerror(title="Erreur", message="Le nom du tube de la
question et celui du tube de la réponse ne doivent pas être
identiques.")
        elif(path.exists(f"./build/temp/{nom}")):
            showerror(title="Erreur", message="Ce tube existe déjà")
        else:
            system(f"mkfifo ./build/temp/{nom}")
```

```
            showinfo(title="Info", message="Le tube est créé avec
succès")


    return logique


def handler_demarrer_simulation(champ_port_serveur :
Entry,etiquette_nombre_client : Label,champ_nom_tube_questions :
Entry,champ_nom_tube_reponses : Entry):

    def logique(event):
        global type_communication,type_socket
        port_serveur = 0
        nombre_client = int(etiquette_nombre_client.cget("text"))
        nom_tube_questions =
f"./build/temp/{champ_nom_tube_questions.get().strip()}"
        nom_tube_reponses =
f"./build/temp/{champ_nom_tube_reponses.get().strip()}"

        if(type_communication=="SOCKET"):
            try:
                port_serveur = int(champ_port_serveur.get().strip())
                if(port_serveur < 1 or port_serveur > 65536 ):
                    showerror(title="Erreur", message="Numero port
invalide (Le numéro du port doit être entre 1..65536) !!")
                    return

                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


                try:
                    s.bind(("127.0.0.1", port_serveur))
                    s.close()
                except socket.error as e:
                    showerror(title="Erreur", message="Ce port est déjà
utilisé !!")
                    return

            except Exception as e:
                showerror(title="Erreur", message="Numero port invalide
!!")
                return
```

```
        else:
            tube_reponses = champ_nom_tube_reponses.get().strip()
            tube_questions = champ_nom_tube_questions.get().strip()
            if(len(tube_questions)==0):
                showerror(title="Erreur", message="Champ du nom de tube
questions est vide !!")
                return
            elif(len(tube_reponses)==0):
                showerror(title="Erreur", message="Champ du nom de tube
réponses est vide !!")
                return
            elif(tube_reponses == tube_questions):
                showerror(title="Erreur", message="Le nom du tube de la
questions et celui du tube de la réponse ne doivent pas être
identiques.")
                return
            elif(not path.exists(f"./build/temp/{tube_questions}")):
                showerror(title="Erreur", message="Le tube de questions
n'existe pas")
                return
            elif(not path.exists(f"./build/temp/{tube_reponses}")):
                showerror(title="Erreur", message="Le tube de réponses
n'existe pas")
                return

        if(type_communication == "SOCKET" and type_socket == "TCP"):
            chemin_client = f"./{chemin_base}/build/client-TCP"
            chemin_serveur = f"./{chemin_base}/build/serveur-TCP"
            commande_serveur = f"{chemin_serveur} {nombre_client}
{port_serveur} > ./build/temp/trace_communicarion_serveur.txt &"
            system(f"{commande_serveur}")

            sleep(1)
            for numero_client in range(1,nombre_client+1):
                commande_client = f"{chemin_client} {numero_client}
{port_serveur} >
./build/temp/trace_communicarion_client_{numero_client}.txt &"
                system(f"{commande_client}")
                sleep(0.025)
                system(f"python3
./cœur_projet/graphique/graphique_client.py {numero_client} &")
```

```
            system(f"python3
./cœur_projet/graphique/graphique_serveur.py &")

        elif(type_communication == "SOCKET" and type_socket == "UDP"):
            chemin_client = f"./{chemin_base}/build/client-UDP"
            chemin_serveur = f"./{chemin_base}/build/serveur-UDP"
            commande_serveur = f"{chemin_serveur} {nombre_client}
{port_serveur} > ./build/temp/trace_communicarion_serveur.txt &"
            system(f"{commande_serveur}")

            sleep(1)
            for numero_client in range(1,nombre_client+1):
                commande_client = f"{chemin_client} {numero_client}
{port_serveur} >
./build/temp/trace_communicarion_client_{numero_client}.txt &"
                system(f"{commande_client}")
                sleep(0.025)
                system(f"python3
./cœur_projet/graphique/graphique_client.py {numero_client} &")

            system(f"python3
./cœur_projet/graphique/graphique_serveur.py &")
        else:
            chemin_client = f"./{chemin_base}/build/client-FIFO"
            chemin_serveur = f"./{chemin_base}/build/serveur-FIFO"
            commande_serveur = f"{chemin_serveur} {nombre_client}
{nom_tube_questions} {nom_tube_reponses} >
./build/temp/trace_communicarion_serveur.txt &"
            system(f"{commande_serveur}")

            sleep(1)
            for numero_client in range(1,nombre_client+1):
                commande_client = f"{chemin_client} {numero_client}
{nom_tube_questions} {nom_tube_reponses} >
./build/temp/trace_communicarion_client_{numero_client}.txt &"
                system(f"{commande_client}")
                sleep(10)
                system(f"python3
./cœur_projet/graphique/graphique_client.py {numero_client} &")
                print("hani 5rejt")
            system(f"python3
./cœur_projet/graphique/graphique_serveur.py &")
```

```
    return logique


def creer_interface(fenetre : Tk,etiquette_nombre_client :
Label,champ_port_serveur : Entry,champ_nom_tube_reponses : Entry) ->
None:
    # Configuration de l'interface
    fenetre.geometry("865x705")
    fenetre.title("Application Principale")
    fenetre.configure(bg="white")
    fenetre.resizable(0,0)

    # Ajout des composants graphiques
    etiquette_nombre_client = Label(text="3", bg="white",
fg="black",font=("Arial",13),pady=0)
    etiquette_nombre_client.place(x=380,y=155)

    champ_port_serveur =
Entry(borderwidth=0,highlightthickness=0,font=("Arial",12),width=17)
    champ_port_serveur.place(x=95,y=520)

    champ_nom_tube_questions =
Entry(borderwidth=0,highlightthickness=0,font=("Arial",12),bg="white",w
idth=17)
    champ_nom_tube_questions.place(x=520,y=415)

    champ_nom_tube_reponses =
Entry(borderwidth=0,highlightthickness=0,font=("Arial",12),bg="white",w
idth=17)
    champ_nom_tube_reponses.place(x=523,y=520)

    boutton_moins = Label(text=" - ", bg="white",
fg="black",font=("Arial",13),padx=0)
    boutton_moins.bind('<Button-1>',
handler_boutton_moins(etiquette_nombre_client))
    boutton_moins.place(x=348,y=155)

    boutton_jouer_test = Label(text=" Démarrer la simulation ",
bg="white", fg="black",font=("Arial",16),padx=0)
    boutton_jouer_test.bind('<Button-1>',
handler_demarrer_simulation(champ_port_serveur,etiquette_nombre_client,
champ_nom_tube_questions,champ_nom_tube_reponses))
    boutton_jouer_test.place(x=345,y=630)
```

```
   boutton_plus = Label(text=" + ", bg="white",
fg="black",font=("Arial",12),pady=0)
   boutton_plus.bind('<Button-1>',
handler_boutton_plus(etiquette_nombre_client))
   boutton_plus.place(x=527,y=155)

   boutton_generer_nombre_client = Label(text="Générer ", bg="white",
fg="black",font=("Arial",13),pady=0)
   boutton_generer_nombre_client.bind('<Button-1>',
handler_generer_nombre_client(etiquette_nombre_client))
   boutton_generer_nombre_client.place(x=412,y=195)

   boutton_generer_port_serveur = Label(text="Générer ", bg="white",
fg="black",font=("Arial",13),pady=0)
   boutton_generer_port_serveur.bind('<Button-1>',
handler_generer_port_serveur(champ_port_serveur))
   boutton_generer_port_serveur.place(x=265,y=520)

   boutton_generer_nom_tube_reponses = Label(text="Générer ",
bg="white", fg="black",font=("Arial",13),pady=0)
   boutton_generer_nom_tube_reponses.bind('<Button-1>',
handler_generer_nom(champ_nom_tube_reponses))
   boutton_generer_nom_tube_reponses.place(x=690,y=519)

   boutton_generer_nom_tube_questions = Label(text="Générer",
bg="white", fg="black",font=("Arial",13),pady=0)
   boutton_generer_nom_tube_questions.bind('<Button-1>',
handler_generer_nom(champ_nom_tube_questions))
   boutton_generer_nom_tube_questions.place(x=690,y=413)

   boutton_verifier_port_serveur = Label(text="Vérifier", bg="white",
fg="black",font=("Arial",13),pady=0)
   boutton_verifier_port_serveur.bind('<Button-1>',
handler_verifier_port_serveur(champ_port_serveur))
   boutton_verifier_port_serveur.place(x=349,y=520)

   boutton_nom_tube_questions = Label(text="Créer   ", bg="white",
fg="black",font=("Arial",13),pady=0)
   boutton_nom_tube_questions.bind('<Button-1>',
handler_creer_tube_nomme(champ_nom_tube_questions,champ_nom_tube_repons
es))
   boutton_nom_tube_questions.place(x=775,y=413)
```

```
   boutton_nom_tube_reponses = Label(text="Créer    ", bg="white",
fg="black",font=("Arial",13),pady=0)
   boutton_nom_tube_reponses.bind('<Button-1>',
handler_creer_tube_nomme(champ_nom_tube_reponses,champ_nom_tube_questio
ns))
   boutton_nom_tube_reponses.place(x=775,y=519)



# Champs de textes
etiquette_nombre_client = None
champ_port_serveur = None
champ_nom_tube_questions = None
champ_nom_tube_reponses = None

type_communication = "SOCKET"
type_socket = "TCP"

# Fênetre
fenetre = Tk()

# L'ajout de 'limage de l'interface
frame = Frame(fenetre, width=865, height=705)
frame.pack()
frame.place(anchor='center', relx=0.5, rely=0.5)
img =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/appli
cation.png"))
label = Label(frame, image = img)
label.pack()

# L'ajout des images de radio
socket_radio_img_2 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_2.png"))
socket_radio_2 = Label(fenetre, image =
socket_radio_img_2,borderwidth=0)
socket_radio_2.place(x=120,y=305)
socket_radio_2.bind('<Button-1>', handler_choix_socket())
socket_radio_img_1 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_1.png"))
```

```
socket_radio_1 = Label(fenetre, image =
socket_radio_img_1,borderwidth=0)
socket_radio_1.place(x=120,y=305)

socket_tcp_radio_img_2 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_2.png"))
socket_tcp_radio_2 = Label(fenetre, image =
socket_tcp_radio_img_2,borderwidth=0)
socket_tcp_radio_2.place(x=200,y=405)
socket_tcp_radio_2.bind('<Button-1>', handler_choix_socket_tcp())
socket_tcp_radio_img_1 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_1.png"))
socket_tcp_radio_1 = Label(fenetre, image =
socket_tcp_radio_img_1,borderwidth=0)
socket_tcp_radio_1.place(x=200,y=405)

socket_udp_radio_img_1 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_1.png"))
socket_udp_radio_1 = Label(fenetre, image =
socket_udp_radio_img_1,borderwidth=0)
socket_udp_radio_1.place(x=360,y=405)
socket_udp_radio_img_2 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_2.png"))
socket_udp_radio_2 = Label(fenetre, image =
socket_udp_radio_img_2,borderwidth=0)
socket_udp_radio_2.place(x=360,y=405)
socket_udp_radio_2.bind('<Button-1>', handler_choix_socket_udp())

fifo_radio_im_1 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_1.png"))
fifo_radio_1 = Label(fenetre, image = fifo_radio_im_1,borderwidth=0)
fifo_radio_1.place(x=610,y=302)
fifo_radio_img_2 =
ImageTk.PhotoImage(Image.open("./cœur_projet/graphique/ressources/boutt
on_radio_2.png"))
fifo_radio_2 = Label(fenetre, image =
socket_udp_radio_img_2,borderwidth=0)
fifo_radio_2.place(x=610,y=302)
```

```
fifo_radio_2.bind('<Button-1>', handler_choix_fifo())


# Création de l'interface
creer_interface(fenetre,etiquette_nombre_client,champ_port_serveur,cham
p_nom_tube_reponses)

# Lancement de l'application
demarrer_application(fenetre=fenetre)
```