✅ RAPPORT : Communication Client/Serveur avec Tubes Nommés et Signaux UNIX

1. Introduction

Ce projet met en œuvre un système de communication entre un **serveur** et plusieurs **clients**, basé sur deux mécanismes fondamentaux du système UNIX :

1. **Les tubes nommés (FIFO)** pour transporter les données.
2. **Les signaux UNIX** pour synchroniser les échanges.

Le client envoie une **question** au serveur en utilisant un tube nommé (`fifo1`).
Le serveur génère ensuite une **réponse**, envoie un **signal SIGUSR1 au client**, puis transmet la réponse dans un deuxième tube (`fifo2`).

L'objectif est d'étudier :

- la synchronisation entre processus,
- le blocage lors de l'ouverture de FIFO,
- l'utilisation de structures en communication,
- la gestion de signaux personnalisés,
- l'organisation d'un projet C en plusieurs fichiers.

2. Architecture du Projet

2.1 Fichiers du projet

Fichier	Rôle
`serv_cli_fifo.h`	Définitions communes (structures, constantes, includes)

`Handlers_Serv.h`	Handlers utilisés par le serveur
`Handlers_Cli.h`	Handlers utilisés par le client
`serveur.c`	Code du serveur, boucle principale, génération des nombres
`client.c`	Code client, envoi de la question et réception de la réponse
`Makefile`	Compilation automatique

2.2 Structures échangées

```
typedef struct {
    pid_t pid_client;
    int nombre;
} Question;

typedef struct {
    pid_t pid_client;
    int taille;
    int valeurs[NMAX];
} Reponse;
```

La communication se fait par **écriture/lecture directe de ces structures** dans les tubes.

3. Fonctionnement du Serveur

Le serveur :

1. Crée les tubes `fifo1` et `fifo2`
2. Installe des handlers :
    - `hand_reveil()` pour SIGUSR1
    - `hand_immortel()` pour bloquer Ctrl+C, Ctrl+Z, etc.
3. Entre dans une boucle infinie :
    - ouvre `fifo1` en lecture
    - lit une structure `Question`

    - génère une liste de nombres aléatoires
    - envoie SIGUSR1 au client
    - écrit la `Reponse` dans `fifo2`

Le serveur ne s'arrête jamais sauf par **kill -9** (modèle "serveur immortel").

4. Fonctionnement du Client

Le client :

1. Installe son handler `hand_reveil()`
2. Demande à l'utilisateur combien de nombres il veut
3. Construit une structure `Question`
4. Écrit cette structure dans `fifo1`
5. Attend un signal SIGUSR1 :
    - via `pause()`
6. Lit la structure `Reponse` dans `fifo2`
7. Affiche la liste reçue

Le signal sert donc de **déblocage intelligent** : le client ne lit pas tant que le serveur n'a pas répondu.

5. Phénomènes Importants

✅ Blocage à l'ouverture des FIFO

- `open(fifo1, O_RDONLY)` bloque **tant qu'aucun client n'ouvre en écriture**.
- `open(fifo1, O_WRONLY)` bloque **tant qu'aucun serveur n'est en lecture**.

> Ce phénomène est central : c'est lui qui garantit la synchronisation naturelle entre client et serveur.

✅ Synchronisation par signaux

Le signal SIGUSR1 sert à :

- informer le client que le serveur a terminé la génération
- permettre au client de sortir de `pause()`

✅ Communication structurelle

Les structures `Question` et `Reponse` sont envoyées **en bloc**, ce qui simplifie l'échange.

✅ 6. Scénario Client / Serveur (image):

Étapes du serveur

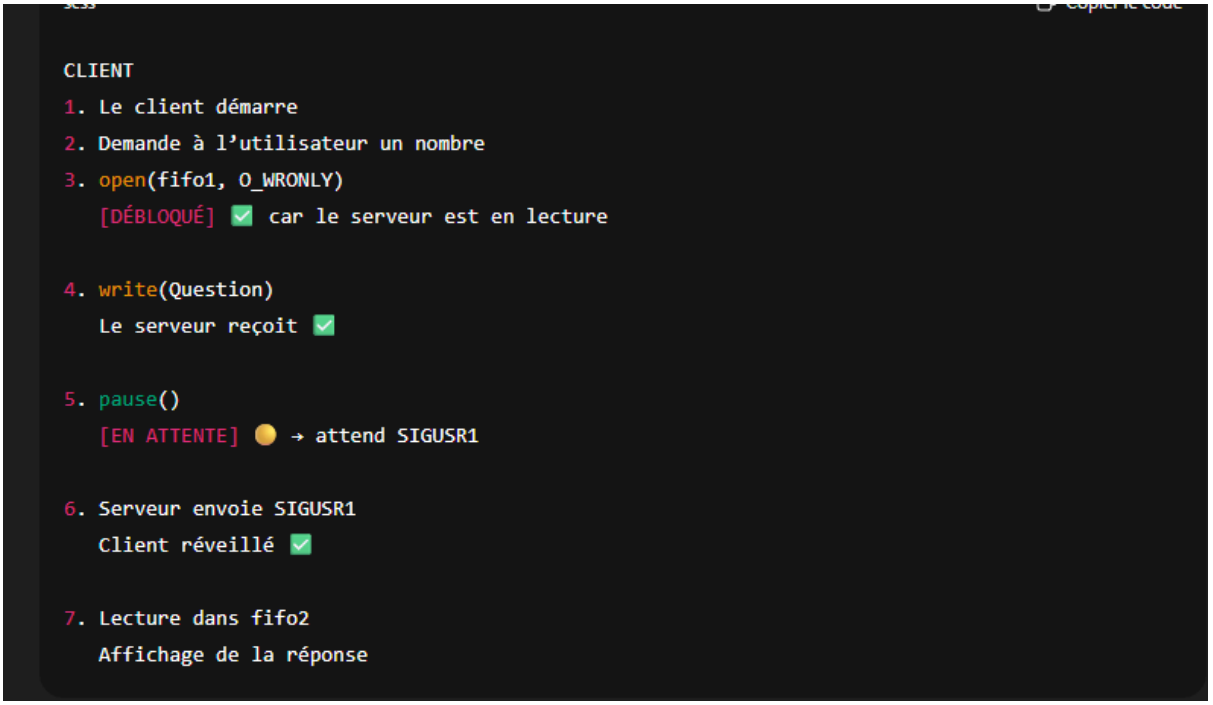
```scss
SERVEUR
1. open(fifo1, O_RDONLY)
   [BLOQUÉ] 🔴  → attend qu'un client ouvre fifo1 en écriture

2. Le client démarre
   Serveur débloqué ✅

3. read() sur fifo1
   [EN ATTENTE] 🟡 → attend la question

4. Le client écrit la Question
   Serveur lit ✅
   Génère les nombres
   kill(pid_client, SIGUSR1) → réveille le client
   Écrit la Reponse dans fifo2
```

Étapes du client



✅ Vision globale :

FIFO → transporte les données
SIGUSR1 → transporte l'information "la donnée est prête"

C'est un système complet de requête/réponse, comme un mini-protocole RPC UNIX.

✅ Conclusion

Ce projet illustre plusieurs concepts essentiels du système UNIX :

- **Communication inter-processus (IPC)**
- **Tubes nommés et leur comportement bloquant**
- **Synchronisation asynchrone via signaux**
- **Structuration d'un projet C en modules `.c` / `.h`**
- **Gestion robuste de processus serveur/clients**

Le serveur joue un rôle central : il répond à chaque demande en générant des valeurs aléatoires et utilise un signal pour notifier le client.
Le client utilise une double synchronisation :

1. **blocage FIFO naturel**
2. **réveil via signal**

Le système obtenu est fiable, modulaire et conforme au modèle “client/serveur”.