Description du projet de messagerie instantanée

Tableau des membres avec les renseignements des élèves présents dans le groupe :

Membres		Numéro de groupe	Numéro étudiants
Zhu	Yulei	Groupe 2	22401396
Ye	Kevin	Groupe 2	
Fan	Maozhen		

Les instructions nécessaires

Objectif : Créer un serveur permettant à plusieurs clients de se connecter et de discuter entre eux via des sockets TCP. Chaque fonctionnalité est associée à une étape précisée dans le sujet.

Instructions d'exécution

Dépendances : uniquement la librairie standard POSIX (gcc, pthread, unistd, arpa/inet, etc.)

Compilation :

1. cd ~/Downloads/L3-Projet-Systeme-main
2. gcc -o serveur server.c -lpthread et gcc -o client client.c -lpthread
3. ./server (premier terminal)
4. ./client (deuxième terminal)
5. ./client (troisième terminal)

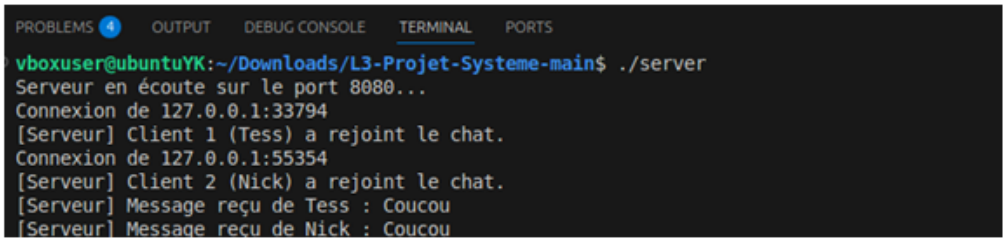
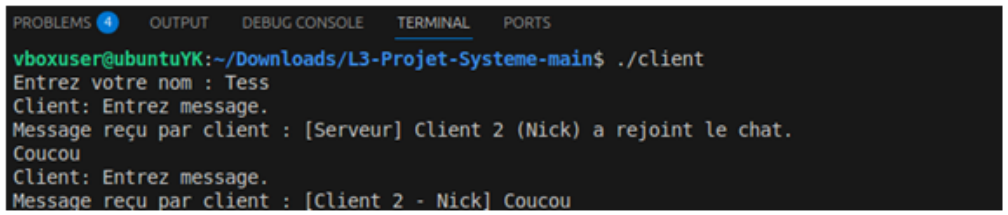
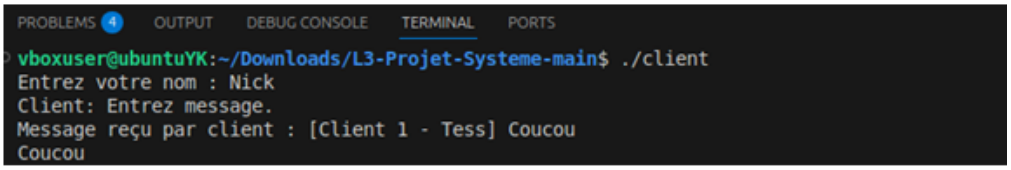
Fonction du code

Étape 1 : Création du serveur

Pour cette première étape, j'ai implémenté un serveur TCP en C qui utilise des sockets pour accepter les connexions entrantes sur le port 8080. À chaque fois qu'un nouveau client se connecte, le serveur crée un nouveau thread avec pthread_create() pour gérer la communication avec ce client.

Chaque thread exécute la fonction traiter_client() qui :

- récupère le nom du client,
- écoute les messages qu'il envoie,
- les diffuse aux autres clients connectés.

	Server 1
	Client 1
	Client 2

Étape 2 – Gestion des messages

Conformément aux consignes, j'ai utilisé une matrice de chaînes de caractères pour stocker les messages par utilisateur :

```
30    char historique_messages[MAX_CLIENTS][MAX_MESSAGES][MSG_LENGTH];
31    int compteur_messages[MAX_CLIENTS] = {0};
```

Chaque ligne correspond à un client, chaque colonne à un message qu'il a envoyé. Un compteur compteur_messages[] permet de suivre le nombre de messages enregistrés pour chaque client.

Étape 3 – Envoi de messages

Chaque client peut saisir un message texte via le terminal. Le message est lu avec fgets() dans le thread envoyer() du client, puis envoyé au serveur via la fonction write().

```
40            fgets(message, BUFFER_SIZE, stdin);
41            message[strcspn(message, "\n")] = '\0';
42
43            write(sclient, message, strlen(message) + 1);
```

Étape 4 – Réception et envoi multiple

Le serveur, lorsqu'il reçoit un message d'un client, le rediffuse à tous les autres clients connectés, sauf à l'émetteur. Cela se fait avec une boucle write() dans traiter_client(), protégée par un mutex pour éviter les accès concurrents.

```
46      pthread_mutex_lock(&mutex);
47      for (int i = 0; i < nb_clients; i++) {
48          if (clients[i].socket != client.socket) {
49              write(clients[i].socket, annonce, strlen(annonce) + 1);
50          }
51      }
52      pthread_mutex_unlock(&mutex);
```

Étape 5 – Affichage des messages

Chaque client a un thread recevoir() qui écoute les messages entrants avec read() et les affiche immédiatement avec printf(). Cela permet de recevoir les messages des autres clients en temps réel sans bloquer la saisie.

```
70      memset(reponse, '\0', BUFFER_SIZE);
71      int n = read(sclient, reponse, BUFFER_SIZE);
72
73      if (n <= 0) break;  // Fin de connexion
```

Étape 6 – Identification

Pour chaque message diffusé, le serveur ajoute l'identité du client à l'aide de l'ID (entouré en rouge) et du nom fourni lors de la connexion.
Exemple de format :

```
Connexion de 127.0.0.1:33794
[Serveur] Client 1 (Tess) a rejoint le chat.
Connexion de 127.0.0.1:55354
[Serveur] Client 2 (Nick) a rejoint le chat.
```

Le nom est reçu dès la connexion du client, et l'ID est assigné automatiquement en fonction de l'ordre d'arrivée.

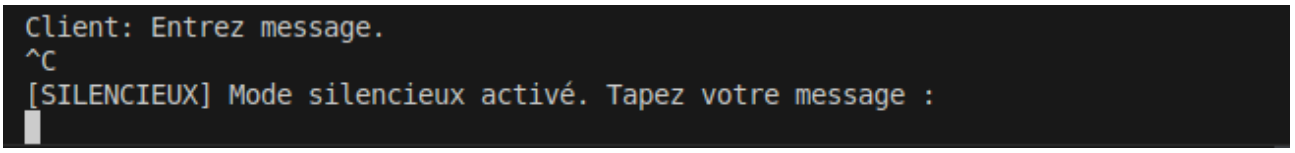
Étape 7 – Blocage d'affichage pour taper un message

Quand un utilisateur appuie sur Ctrl+C, un mode silencieux est activé côté client. Cela est géré via un signal(SIGINT, handler).
Pendant ce mode :

- Tous les messages reçus sont stockés dans un tampon local.
- Aucune interruption visuelle ne gêne la rédaction du message.
- Une fois le message envoyé, le client affiche tous les messages reçus pendant l'interruption.

Cela permet à l'utilisateur d'écrire tranquillement sans perdre d'informations.

Pour essayer cela, il faut créer un server puis 2 clients faire ctrl + c qui affichera le message suivant :



Et pendant que l’autre utilisateur parle, tous les messages seront stockés et renvoyés lorsque l'utilisateur aura écrit un message.