Petit Bilan de tout ce bordel.

1. Classe Socket et structure Paquet

Tout d'abord, tu as la classe **Socket**, qui est commune au client et au serveur. C'est elle qui fait le lien entre les deux et definit certaines choses, comme la structure Paquet, pour qu'ils s'attendent tous les deux a recevoir la même structure, de la même taille.

Le paquet que le client et le serveur s'échangent, c'est la structure PerudoPaquet, définie dans <Socket.h> ligne 40. Pour l'instant, il n'y a que le message (de 20 caractères) mais tu peux rajouter ce que tu veux, tant que c'est du C. (les vecteurs, templates et classes ne passent pas dans des sockets, il connait pas).

Les deux méthodes définies dans cette classe (Socket), c'est remplirStructure(), qui va demander au joueur de rentrer son message et qui va le mettre dans le paquet, et afficherPaquet() qui va afficher le contenu du paquet.
Évidemment, quand tu va rajouter quelque chose dans la structure Paquet va falloir modifier ces deux fonctions en conséquence.

2. Classe Client

Il y a ensuite la classe **Client** **qui hérite de Socket (autrement dit, toutes les méthodes et les attributs de Sockets sont aussi dans Client)**, qui va se charger de se connecter a un serveur (par défaut a l'adresse 127.0.0.1) (défini a la ligne 31 de Client.cpp), et d'envoyer et de recevoir des paquets.

4 méthodes :

- initialiser(), qui va configurer la connexion et va apeller remplirStructure() sur monPaquet, c'est le fameux Paquet du client, défini comme un attribut de la classe (Client.h ligne 20). Ca va donc demander au client de rentrer son message et le stocker dans le paquet (en l'occurence, ici, ca va le stocker dans monPaquet).

- Envoyer() SSI la connexion est établie (ligne 42), ca va envoyer monPaquet (le paquet du client) au serveur (ligne 48)

- recevoir() va attendre que le serveur lui envoie quelque chose, et remplacer monPaquet par le paquet recu.

- Fermer() ferme la socket proprement.

3. Classe Serveur

Après, et pour finir d'ailleurs, la classe **Serveur**, qui hérite également de Socket.

- amorcer() va démarrer le serveur et écouter, pour choper chaque connexion entrante (des qu'un client se connecte).
- RecevoirPaquet() c'est **LA** méthode subtile, donc je vais essayer d'être clair.
    - Tout d'abord, on entre dans une boucle, pour que plusieurs clients puissent se connecter. Le fait qu'il y ait une boucle dans cette méthode signifie qu'il n'y a pas besoin de faire un while(quelquechose) { serveur.recevoirPaquet() ; } dans le main, ca le fait déjà.
    - Ensuite, on crée un nouveau Sclient. Qu'est ce que c'est que ca ? C'est simplement une structure (et oui, encore une !) qui va contenir l'identifiant du socket du client qui va se connecter au serveur, l'identifiant du thread qu'on va utiliser pour lui et son paquet (et oui, qui dit plusieurs clients dit plusieurs paquets a gérer). Les autres variables ne sont pas intéressantes, elles sont pour la gestion des sockets. (voir Socket.h ligne 43) Tu noteras qu'on le crée dynamiquement, parce qu'on ne sait pas combien il va y avoir de client, donc on en crée un pour chaque client qui se connecte.
    - Si tout va bien (c'est ca que veut dire « sock_err != SOCKET_ERROR »)
    - On crée un nouveau thread qui va utiliser le fameux identifiant de thread du client (contenu dans le Sclient qu'on a crée) qui s'est connecté pour lancer la fonction newClient (définie dans Socket.h ligne 58) avec la structure Sclient du client en paramètre.
    - Le thread attend la réception d'un paquet et affiche son contenu des qu'il le reçoit.
    - Si le msg = « fin » alors on arrête le thread (mais pas le programme)
    - Et on ferme la socket du client concerné.
    - Derrière tout ca, la méthode recevoirPaquet() est dans une boucle (je le rappelle), ca veut dire qu'une fois qu'elle aura lancé le thread, elle re-boucle, recrée un Sclient quand quelqu'un se connecte et crée un nouveau thread. C'est ce procédé qui permet d'avoir un serveur multithread qui est connecté a plusieurs clients.
- La methode fermer(), ferme le socket du serveur. Proprement.

Voilà, j'admets que c'est pas évident, mais ca marche comme ca :D

IMPORTANT :

la classe Socket doit être identique pour le serveur et le client, il faut donc que les fichiers Socket.h et Socket.cpp, qui sont a la fois dans rep client et dans rep serveur soient identiques. Quand vous en modifiez un, n'oubliez pas de modifier l'autre (ou de le copier).

NB :

Pour compiler le client, place toi dans le répertoire Part Client et cmd :

g++ -g -o client Client.cpp main.cpp Socket.cpp

Pour compiler le serveur, répertoire Part Serveur et cmd :

g++ -g -o serv Serveur.cpp main.cpp Socket.cpp -pthread