Travail pratique #1

Objectif : Pratiquer votre programmation des sockets Berkeley en C++ 17

Consignes :

- Votre programme doit être développé en C++ 17
- Votre programme doit utiliser le « build-system » CMake.
    - Si possible utilisez la version de visual studio 2019 qui ajoute les fonctionnalités Cmake nativement
- Le travail se fait en équipe de **trois** personnes
    - Pas deux, ni une
    - On est 33 donc théoriquement, dans une algèbre standard, c'est un multiple de 3
- Si le TP comporte des questions à développement, y répondre dans un README au format Markdown à la racine de votre Git
- Le rendu du TP se fera au moyen du gestionnaire de version Git
    - Un rappel de l'outil Git sera fait en classe
    - **Attention** : le graphe des contributeurs au projet peut être un excellent indicateur de qui à travailler et avec quelle proportion sur le projet
        - **Non un TP n'a pas besoin d'un chef de projet qui fait de la gestion**
- La date de rendu du TP est le **3 octobre 2021 à 23H59**
- Il vous est **INTERDIT** d'utiliser une librairie faisant le réseau, le multithreading ou l'asynchrone pour vous (à l'exception de WinSock2 et socket évidemment)
- **Rappels du plan de cours** :
    - Tout travail remis en retard, et ce, sans motif valable, sera évalué sur 50%. Les retards pour les travaux sont notés à raison de -10% par jour (p. ex. pour 2 jours de retard donnera -20% de la note sur 50% de l'évaluation).
    - Tout travail ne compilant pas sera noté sur 20% seulement
    - Un malus allant jusque -20% sanctionneront tous les travaux dont le français sera jugé incorrect.
- **Zone risquée :**
    - Si vous avez des étapes avant la compilation de votre projet veuillez m'indiquer les démarches à suivre dans un README au format Markdown à la racine de votre Git.

Énoncé:

1er partie :

Vous êtes l'équipe réseau d'une entreprise de jeu vidéo. Votre producteur veut que le prochain jeu du studio soit en multiplayer. Malheureusement vous êtes les seuls développeurs réseaux de l'entreprise et donc les seuls qui puissent utiliser ces belles petites sockets. Vous allez donc devoir simplifier leurs utilisations pour le reste de votre équipe.

Conseils :

La liste n'est pas exhaustive, il faudra rajouter différents éléments qui pourrait manquer ou non pour réaliser cette librairie.

- Classe à avoir :
    - Connection
        - C'est via elle que vous envoyez et recevez vos messages
    - Terminal
        - C'est ce qui permet au serveur de créer une connexion lorsqu'il reçoit une demande de connexion d'un client.
    - Une classe mère
        - Deux fonctions importantes :
            - (Server) Listen
            - (Client) Connect
        - Paramètre minimum :
            - Protocol, ip, port :
                - Exemple « udp//127.0.0.1 :42666 »
            - Une config (struct) pour les callbacks (points suivants)
    - Vous devez gérer UDP & TCP :
        - Donc faites une classe UDPConnection et TCPConnection qui hérite de Connection
- Soyez Asynchrone :
    - Utilisez les callback (std ::function) montré au cours 2 :
        - Lorsque qu'une personne se connecte sur le serveur
            - Il faut donner au serveur une Connection pour qu'il puisse envoyer des messages au client
        - Lorsque le client est bien connecté au serveur
            - Il faut donner la Connection au client aussi
        - Lorsque vous recevez un message
            - Pensez à toujours identifier sa provenance dans les paramètres de la callback
        - Lors d'une déconnexion
    - Il vous faudra donc 2 threads :
        - Le principal
        - Celui pour les fonctions réseaux (via une methode Update)
    - Réseaux non bloquant évidement 😊 (Partie 2 du cours sur les sockets)

- Votre programme doit compiler
- Votre programme ne doit jamais crash (attention aux déconnexions)

2ème partie :

Il faut maintenant tester votre librairie pour cela votre producteur vous demande un petit système de chat client-serveur simple. (*J'insiste sur le simple*)

Conseils :

- **Serveur :**
  - L'exécutable doit écouter sur une adresse passé en argument.
  - L'exécutable du serveur doit gérer les connexions et les stockés.
  - Broadcast les messages à tous les utilisateurs (sauf l'émetteur évidemment)
- **Client :**
  - L'exécutable doit se connecter au serveur passé en argument.
  - L'exécutable doit lire et recevoir les messages
  - Essayer de fournir un petit effort sur l'affichage ^^ (en ligne de commande, je ne vous demande pas de refaire discord évidement) (non essentiel)

Question(s) :

Quel protocole choisi-t-on pour le chat du jeu vidéo ? UDP ou TCP et pourquoi ?