.

Projet FAR 2021-2022 : une application de messagerie instantanée

1. Organisation des groupes et Livrables :

Nombre de groupes : 20 maximum (binômes ou trinômes, pas plus. Pas de monômes)
Pour équilibrer les groupes, chacun doit être constitué d'élèves issus de parcours pré-IG différents (mélange DUT et BTS d'un côté et Prépa et Licence de l'autre).

Chaque livrable est à retourner sur Moodle à la **fin du Sprint** (les dimanches soirs de la fin de chaque sprint, à 23h55 au plus tard)

Tout dépôt en retard est pénalisé (en fonction du nombre de jours ou d'heures de retard).

Un livrable est constitué des éléments suivants :
1. code source bien structuré (décomposer le code en fonctions de petite taille, utiliser des variables globales que lorsque c'est nécessaire, utiliser des constantes à volonté, ...), lisible (identifiants parlants pour les variables, constantes et fonctions) et bien commenté,
2. document PDF décrivant :
    i) le protocole de communication entre les clients et le(s) serveurs (diagramme de séquence UML),
    ii) le cas échéant (si le protocole ne suffit pas), l'architecture de l'application (en termes de processus, threads, ressources/données partagées, mécanismes de synchronisation utilisés, ...),
    iii) les difficultés rencontrées, les choix faits pour les surmonter, et les problèmes qui persistent,
    iv) la répartition du travail entre les membres du groupe (très important, pour montrer la bonne organisation de l'équipe),
    v) comment compiler et exécuter le code (commandes gcc, make, ...).

La note finale du projet : moyenne pondérée des notes de livrables et de la note de l'oral de fin de projet.

Du fait qu'un seul sujet est proposé à toute la promo, une attention particulière sera portée sur le plagiat (au sein de la promo et vis-à-vis du code disponible sur Internet). Un logiciel anti-plagiat (Compilatio.net) sera utilisé pour analyser et comparer les rendus. Tout code copié doit être explicitement mentionné, sinon de lourdes pénalités seront appliquées.

2. Organisation du travail

---

* Programmation en langage C :

Sprint 1 : Semaine du 29.03. 3 séance encadrée : Rendu le 17/04
Partie 1 :

Sujet (séance 1) : Un serveur relaie des messages textuels entre deux clients
1 programme serveur et 1 programme client, lancé deux fois (deux processus distincts)
1 seul processus/thread serveur pour gérer les 2 clients, qui envoient leurs messages à tour de rôle (client 1 : write puis read, et client 2 : read puis write)
l'échange de messages s'arrête lorsque l'un des clients envoie le message « fin ».
Ceci n'arrête pas le serveur, qui peut attendre la connexion d'autres clients.
On utilisera le protocole TCP.

Sujet (séance 2 et 3) : Un serveur et un client multi-threadés

v1 : Utiliser le multi-threading pour gérer l'envoi de messages dans n'importe quel ordre (on ne sera plus contraint par le fait que c'est le premier client qui se connecte qui envoie un message en premier et attend que l'autre lui écrive un message pour pouvoir envoyer un autre message) :
programme client : 1 processus pour la saisie (avec fgets) et l'envoi du message au serveur
et 1 autre processus pour la réception des messages du serveur et leur affichage (avec puts)
soit le programme principal plus un thread minimum

programme serveur : 1 thread pour relayer des messages du client 1 vers le client 2 et 1 autre pour relayer les messages du client 2 vers le client 1,
pour faciliter le passage en v2 on pourra appeler la même fonction en créant le thread en passant en argument le descripteur et en interrogeant le tableau des clients

v2 : relayer des messages entre n clients dont les descripteurs de sockets de connexion sont stockés dans un tableau (rappel : les descripteurs clients peuvent être récupérés via la fonction accept) .
Côté serveur : 1 thread par client pour écouter les messages qui proviennent du client et les diffuser vers tous les autres clients.
Côté client : rien ne change par rapport à la v1 de cette itération.

Sprint 2 : Semaines du 18 au 30 avril. 4 séances encadrées :

Semaine 1:

1. Définir un protocole pour les commandes particulières envoyées en messages depuis le client(exemple @... sur discord)

2. Création d'une fonction message privé, un client peut choisir d'envoyer à un autre client en particulier :

a. première version avec utilisation du numéro de client ou un identifiant
b. ajout d'un pseudo lors de la connexion d'un client et utilisation du pseudo pour les échanges privé
c. Gestion d'erreur coté serveur sur l'existence du destinataire

3. Création d'une fonction de déconnexion, une commande permet au client de se déconnecter, le serveur, enlève le client de la liste, clôt alors la connexion (shutdown ou close) puis finit le thread associé.

4. Amélioration du code et gestion des nouveaux clients.
   - Si pas déjà fait, ajout d'un mutex pour le tableau des clients.
   - Ajout d'un sémaphore indiquant le nombre de place restante sur le serveur pour faciliter le remplacement de client (Utiliser la bibliothèque sys/sem.h : exemple ici !)
   - Ajout d'une variable partagée pour une fermeture propre des threads lors de la déconnexion des clients et la connexion de nouveau clients