Laboratoire traitant sur la programmation réseau.

Objectifs – Familiariser l'étudiant à la programmation réseau
Amener l'étudiant à développer une application utilisation un protocole de communication simple qui aura été conçu par celui-ci

Date de remise

Remise ***AVANT*** le 20 ***Février à 11h59 (midi)***

Remise

Sur ***1 seul fichier archive*** (ZIP, 7z, rar, …) sur ***Moodle*** contenant :

- Un fichier Word dans lequel on retrouve votre nom ainsi qu'une brève description de comment utiliser vos programmes (*un genre de README*)
- Nommer adéquatement votre archive. Exemple : CodePermanent_Lab3.zip
- Nommer adéquatement vos fichiers en général
- La ou les solutions/projets pour votre/vos programme(s)
  - *De préférence utiliser Visual Studio 2015 ou plus récent pour votre implémentation*

Directives générales

- Ce travail doit être réalisé ***seul***
- Il est important de rappeler que le plagiat, même partiel, est interdit.
- Le développement doit être effectué en **C/C++** et fonctionner **sous** le système d'exploitation **Windows**

Critères d'évaluation :

- Respect des consignes et des directives
- Qualité, stabilité et fonctionnalité des implémentations

Auteurs et collaborateur du laboratoire :

- Daniel Audet (*Auteur*)
- Louis-André Guérin (Collaborateur)

Programmation d'une application Client(s)-Serveur

80% du travail (*points*) est de réussir à effectuer adéquatement la communication entre un client et un serveur. La communication devra permettre de transmettre de manière coordonnée de l'information (*des fichiers*).

20% du travail (*points*) est de permettre à votre serveur de gérer la connexion de plusieurs clients simultanément (*Exemple : 3,4 ou infini*).

La logique de votre « protocole de communication » sera la même que ce soit avec 1 seul client ou clients multiple. C'est simplement la portion de la gestion des connexions qui sera différente pour les connexions multiples.

Protocole

- Vous devrez « concevoir » une séquence de communication afin que les différents programmes puissent échanger de l'information de manière ordonnée. Vous créerez donc en se faisant, ***un protocole***.

Protocole

- Vous devrez « concevoir » une séquence de communication (*et d'états*) afin que les différents programmes puissent échanger « de l'information » de manière ordonnée. Vous créerez donc en se faisant, ***« un protocole »***.
- Il est fortement suggéré de **formaliser votre séquence de communication** ainsi que des **états possibles** lors des différents cas de figure lors de l'exécution des programmes

Informations supplémentaires sur le fonctionnement des programmes : Serveur

- Dès son ouverture, le serveur sera en attente d'une tentative de connexion client
- Lorsqu'un client tente une connexion, le serveur enclenche une séquence de validations avec le client afin d'accepter la connexion
  - Il devrait y avoir des échanges afin de valider et confirmer la connexion
  - Le client devrait fournir des codes d'accès « *crédentials* » afin de confirmer au serveur qu'il a effectivement l'autorisation de se connecter
    - Même si dans la réalité on ne fait JAMAIS ça, vous pouvez « hardcoder » lesdits codes directement dans vos programmes afin de faciliter, simplifier et accélérer cette gestion
    - Tous les clients peuvent utiliser les mêmes codes de connexion ou pas

- ❖ Dans le cas d'un serveur permettant les connexions simultanées multiples, il devrait se remettre en attente d'une autre connexion client dès que la connexion sera effectuée
  - Sinon, il devra attendre que la connexion client soit terminée avant de se remettre en état d'attente de connexion.
- ❖ Une fois la connexion établie, le serveur enverra une liste de fichiers disponible au client et se placera en attente d'une commande de la part du client
  - Vous pouvez assumer qu'il n'y aura qu'un seul répertoire contenant des fichiers et ce répertoire peut avoir un nom pré-spécifié dans votre code
  - Le contenu (*la liste des fichiers présent*) dudit répertoire devra cependant être chargé « dynamiquement » en fonction du contenu de celui-ci

➢ Le serveur doit gérer la déconnexion (*demande de déconnexion de la part*) d'un client

- ❖ Dans le cas d'un serveur à connexion unique, le serveur doit se remettre en attente de connexion après une déconnexion

Informations supplémentaires sur le fonctionnement des programmes : Client

- ➢ Le client doit pouvoir faire parvenir une « commande » au serveur pour lui indiquer lequel des fichiers disponibles il désire télécharger
- ➢ Le client doit pouvoir redemander la liste des fichiers disponibles au serveur
- ➢ Le client doit pouvoir « effectuer une déconnexion » du serveur
  - ❖ Cette déconnexion doit se gérer adéquatement et ne pas occasionner de « crash » de part ou d'autre de la connexion
  - ❖ Le serveur doit se remettre en mode « attente de connexion » par la suite

Transfert d'information – Client-Serveur

- ➢ Les informations échangées entre le client et le serveur seront de différents types
  - ❖ Vous devrez donc vous assurer d'implémenter, dans le contenu de vos échanges et dans la structure de vos programmes, le nécessaire afin de distinguer le type et donc le contenu desdites informations

- Liste non exhaustive mais pour donner une idée :

  Demande de connexion, validation des accès, confirmation de connexion, erreur de connexion, demande de déconnexion, confirmation de déconnexion, envoi de la liste des fichiers disponibles, demande de retourner la liste, demande de téléchargement d'un fichier, contenu partiel ou complet d'un fichier (*est-ce que le transfert est terminé ?*), …

Validation et éléments à valider pour assurer un bon fonctionnement de vos programmes

- Les fichiers échangés doivent être fonctionnels après le transfert
  - Effectuer des tests de transfert avec des fichiers de différents types :

    BMP, JPG, MP3, txt, exe, … pour permettre une validation minimale sur le fonctionnement de vos transferts.
- Valider les différents cas de figure dans la séquence
  - Erreur de code d'accès pour s'assurer que le serveur gèrera correctement
  - Demande de la liste de la part d'un client
  - Demande d'un fichier inexistant ou demande invalide
  - Gestion de la réception d'une commande invalide
  - Etc…
- *Attention à ne pas utiliser une librairie qui ferait tout le travail à votre place*

Informations

- Vous pouvez différentes mécaniques pour implémenter vos programmes
  - Socket (*attention synchrone vs asynchrone*), TCPListener, …
  - *Attention à ne pas utiliser une librairie qui ferait tout le travail à votre place*

- Liens pertinents avec exemples simples :
  - https://www.geeksforgeeks.org/socket-programming-cc/
  - https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/
  - https://riptutorial.com/cplusplus/example/23999/hello-tcp-server
  - https://www.linuxhowtos.org/C_C++/socket.htm