Rapport laboratoire n°6

-

Réseaux d’ordinateurs - 6GEN723

-

BALJ1708609 – Baldo Jérôme

Table des matières

I. Introduction

Actuellement je n'ai pas pu réaliser l'option de cacher la console ou de faire le service Windows. À la suite nombreux bug, je n'ai pas pu vous faire le chiffrement.

Le serveur est mis sur une adresse statique 127.0.0.1 au port 8080. Les commandes fonctionnent ainsi que le téléchargement / chargements.

Mon dossier est constitué d'un dossier exécutable pour effectuer la démonstration, d'un dossier contenant les 2 projets visual studio et d'un dossier comprenant le rapport ainsi que la vidéo.

II. Présentation du laboratoire

1. Architecture et tour d'horizon des méthodes

Pour la partie pirate :

Voici les classes utilisées pour le fonctionnement :

- Main.cpp =>
  - Objectif : faire rouler l'application côté pirate. Il permet de créer un socket qui se connecte au serveur. Il dialogue ensuite avec celui-ci en jouant la réception et les envois de message avec le serveur
  - Méthodes utilisées :
    - Aucune
- SocketPirate =>
  - Objectif : a pour objectif de proposer au main un outil permettant de communiquer avec le serveur. Il gère aussi la réception et l'envoi pour les entêtes et la réception des fichiers.
  - Méthodes utilisées :
    - ReceivFichier => pour réceptionner un fichier
    - GetPort => retourne le numéro de port
    - GetIp => retourne l'adresse IP
    - Receive => réception d'un entête
    - Send => envoi d'un fichier ou d'un entête
    - Connect => permet la connexion du socket vierge avec le serveur
    - RecupPort => permet de récupérer le numéro de port
    - RecupIP => permet de récupérer l'adresse IP
    - InitialisPirate => permet de récupérer IP et port (méthodes Recup) et de connecter le socket au serveur (Connect).
    - ~Socket => destructeur qui spécifie de fermer le socket et de nettoyer la mémoire
    - Socket() => constructeur permettant d'initialiser un socket vierge (en attente de InitialisPirate)
- Dialogue.h =>
  - Objectif : cette classe permet de faire l'interface et conserver les chemins pour les dossiers de téléchargement et chargement.

- Méthodes utilisées :
  - Dialogue => Il met à nul les attributs repertoireChargement et repertoireTelechargement.
  - ChoixOption => interface permettant de choisir l'option à faire sur le serveur. Il retourne un numéro de choix.
  - OptionEnvoi => en prenant le numéro de choix générer la méthode de choix Option, utilise un switch pour préparer l'entête à envoyer au serveur
  - InitRepertoires => méthode qui permet de créer les chemins de répertoire de chargement et de téléchargement
  - GetRepertoireChargement => retourne le chemin du répertoire de chargement
  - GetRepertoireTelechargement => retourne le chemin du répertoire de téléchargement

Pour la partie Serveur :

Voici les classes utilisées pour le fonctionnement :

- Main.cpp
  - Objectif : gestion du socket serveur et dialogue avec le client
  - Méthodes utilisées :
    - ActiviClient => prend en entrée le socket client nouvellement crée. Il utilise une boucle pour continuellement dialoguer avec le client jusqu'à la déconnexion. Dans la boucle il traite la réception de l'entête. Selon l'entête, il reverra un entête avec les informations cibles ou le fichier demandés. Il sortira de sa boucle quand le client le notifiera.
    - Main => créer le socket du serveur, le bind et le met en position d'écoute. Une fois réalisé, il attend un client dans une boucle. Quand le client est accepté, celui-ci envoyé dans la méthode ActivClient.
- Commande.h
  - Objectif : permet de réaliser une commande système sur l'ordinateur. Il est en charge de retourner au main le résultat de cette commande.
  - Méthodes utilisées :
    - ExecCommand() => prend en entrée la commande ainsi que la localisation dernier répertoire courant. Il va exécuter la commande et retourner dan une string son résultat au main.

- SocketClient.h
  - Objectif : il stocke le socket du client cible et gère la communication avec le client
  - Méthodes utilisées :
    - Socket() => avec le socket client initialise l'attribut
    - ~Socket => destructeur qui permet de fermer l'objet.
    - Receive => réception d'un entête
    - Send => envoi d'un fichier ou d'un entête
    - ReceivFichier => pour réceptionner un fichier
- SocketServeur.h
  - Objectif : Créer un socket serveur , le binder, le mettre sur écoute mais accepter un client
  - Méthodes utilisées :
    - SocketServeur() => constructeur initialise le port, l'IP et le socket serveur

- ~SocketServeur() => le destructeur permet de fermer le socket serveur et nettoyer la mémoire.
- Listen() => il crée le socket serveur ensuite le bind, et le met sur écoute
- Accept() => permet d'accepter le client et de retourner le socket client créé

2. Bibliothèques utilisées et syntaxe particulière

Les bibliothèques utilisées

- <winsock2.h> - Il fournit des fonctions pour créer des sockets et pour communiquer via le protocole TCP/IP.
- <ws2tcpip.h> - Cette bibliothèque fournit des fonctions pour la gestion des adresses IP et des noms de domaine.
- <iostream> - Cette bibliothèque fournit des fonctions pour la lecture et l'écriture de flux de données sur la console.
- <string> - Cette bibliothèque fournit des fonctions pour la manipulation de chaînes de caractères.
- <vector> - Cette bibliothèque fournit une structure de données de vecteur dynamique, qui permet d'ajouter et de supprimer des éléments à partir de la fin du vecteur.
- <filesystem> - Cette bibliothèque fournit des fonctions pour la manipulation de fichiers et de répertoires.
- <fstream> - Cette bibliothèque fournit des fonctions pour la lecture et l'écriture de fichiers.
- <cstdio> - : Il fournit des fonctions pour la manipulation des entrées/sorties standard.
- <sstream> : Cette bibliothèque fournit des fonctions pour la manipulation des chaînes de caractères sous forme de flux.
- <iomanip> : Cette bibliothèque fournit des fonctions pour la manipulation de l'affichage en console, telles que la mise en forme de nombres et de dates.

La syntaxe particulière

Voici les syntaxes particulières qui m'ont servi pour le laboratoire :

- Inet_pton => conversion d'une IPV4 en structure binaire
- Ofstream => flux de sortie vers un fichier
- Ios ::out => ouverture en mode écriture
- Ios ::binary => ouverture en mode binaire
- WSACleanup => ferme le socket
- Getline => lecture d'une ligne entière
- WSAStartup => initialiser socket
- MAKEWORD => spécification de version de socket
- Reinterpret_cast => convertir un pointeur d'un type à un autre
- Sockaddr_in => structure pour adresse IPv4 et son port associé
- Sockaddr => structure d'adresse générique pour les sockets
- Filesystem ::path et filesystem ::current_path => gestion des répertoires et fichiers
- _popen et _pclose => lecture d'un flux et récupération du flux
- feof => fin d'un flux
- fgets => Lecture d'une ligne depuis un flux d'entrée et stockage dans un buffer
- ifstream fileIn => utilisé pour lire des données depuis un fichier (flux de données)
- freeConsole => ferme la console associée à une application Windows.

III. Exécution du laboratoire

Vous pouvez naviguer dans le dossier executables. Il comprend le dossier pirate et serveur. Chacun possède un exécutable.

Pour le pirate vous avez un dossier fichier pirate pouvant vous servir pour le chargement.

Pour le serveur vous avez un dossier dossier_recuperer avec des fichiers à télécharger.

Le téléchargement et le chargement possèdent un fonctionnement particulier. Pour télécharger, il faut vous situer dans le répertoire courant souhaité du serveur. En tapant 2, vous donnerez le nom du fichier avec son extension (ex : test.txt). Celui-ci sera automatiquement télécharger dans le dossier de téléchargement pirate situé du côté client.

Pour charger un fichier spécifique sur le serveur, vous devez vous positionner sur le répertoire courant souhaité. Par la suite vous devez réaliser un copier-coller du fichier souhaité dans le dossier chargement pirate côté client. Enfin vous devez 3 en option pour charger le fichier en indiquant le nom et son extension (ex : test.txt). Le client et le serveur feront le reste pour charger le fichier.

Pensez bien à donner votre option dans le menu contextuel avant de réaliser votre action (commande, fichier). De plus il y a une erreur bloquant la console côté client quand vous tapez autre chose que les chiffres proposés.

IV. Explication des points clés

Les sockets

Dans un souci de clarté pour mon code, j'ai souhaité le mettre en programmation objet. Mon laboratoire n°3 ne fonctionnait pas en objet. Je me suis donc rabattu sur une solution utilisant le sockaddr_in. Cette utilisation permet d'utiliser des adresses IPV4 ou IPV6 en TCP. La seule contrainte est de caster en sockaddr alors de l'élaboration du socket. Pour cela j'ai choisi d'utiliser reintepreter_cast qui va caster un pointeur d'un type en un autre type de pointeur.

L'exécution des commandes

L'exécution m'a donné des difficultés. La fonction system déclenche la commande mais ne prend pas en compte le résultat de la commande. La problématique était donc de déclencher la commande et de réceptionner le résultat. Après recherche j'ai trouvé la fonction _popen et _pclose.

L'autre problématique est le répertoire courant du serveur. Après chaque commande, la localisation du répertoire revenait à son état initial. Ainsi il fallait conserver la dernière position. Ainsi après chaque commande, la dernière position est stockée dans une variable. Pour chaque lancement d'une commande, il y a une commande combinée :

- Cd avec la dernière position

- La commande à exécuter
- Cd (pour avoir la position après exécution)

Le résultat est récupéré et la dernière ligne est stocké dans la variable (car reprend le dernier cd).

Le système d'entêtes

Le système des entêtes permet au serveur de choisir la fonctionnalité souhaitée. L'entête est petit en taille et ne nécessite qu'un simple send (idem pour la réception). L'avantage de l'entête est qu'il peut avoir des informations en plus que le type de fonctionnalités à déclencher (ex : CHARGEMENT\\test.txt\\1475\\ => demande de chargement sur le serveur avec un fichier se nommant test d'extension txt pour une taille de 1475)

Le téléchargement de fichiers

Le téléchargement des fichiers se réalisent en prenant en compte 2 envois (que ce soit pour le client ou le serveur).

La difficulté était que le destinataire attendait infiniment lorsque je ne prenais la méthode de réception des entêtes. Je me suis rendu compte que le destinataire ne recevait pas tous les paquets bien que la source notifie de l'envoie totale. Ainsi j'en ai déduit qu'il fallait que le destinataire reçoive la taille du fichier avant et créer une boucle de réception. Pour cela j'ai réalisé une valeur byte reçu qui incrémente à chaque itération les bytes reçus. Tant que les bytes reçus ne sont pas égales à la taille du fichier alors la boucle continue. Par ailleurs il y avait la problématique du buffer accueillant les paquets. La méthode recv prend en argument le pointeur du contenant des paquets. Il fallait donc aussi incrémenter ce pointeur pour chaque byte reçue. Aussi la fonction recv prend en argument la taille du paquet attendu. Ainsi il fallait décrémenter la taille totale du fichier avec chaque byte reçue.

Le format du fichier à télécharger n'est pas important car je me suis concentré non pas sur le contenu du fichier mais sur sa représentation en bytes.

Le serveur toujours en écoute

Le serveur crée son socket d'écoute. Il instancie une boucle permettant de toujours récupérer un client. Quand celui-ci est accepté alors il est isolé dans la fonction ActivClient qui utilise aussi une boucle pour le dialogue tant que client ne notifie pas la déconnexion. Cette implémentation permettrait du multithreading.

L'architecture en objet

J'ai forcé pour faire une architecture en objet pour le pirate et le serveur. Cela a dû me faire changer la structure du socket (en passant sur sockaddr_in). Maintenant l'intérêt de cette structure est d'ajouter de la clarté au code mais aussi de permettre une meilleure évolutivité.

V. Points d'améliorations

- Masquer la fenêtre de la console ou ne pas l'ouvrir tout simplement
- Rendre l'application serveur dynamique en utilisant automatiquement l'IP de l'ordinateur (dans le cas où ma cible serait un particulier ayant une IP dynamique)
- Faire un chiffrement qui fonctionne malgré la navigation
- Revoir la méthode des commandes et passer par une bibliothèque externe pour simplifier.
- Faire du multithreading pour accueillir plusieurs clients. Cela permettrait de faire plusieurs actions sur le serveur en même temps (un peut faire le téléchargement de fichiers, un autre le chargement, un autre les commandes, etc).
- Corriger l'erreur du menu contextuel
- Améliorer l'affichage du côté client pour les réponses du serveur.
- Mettre l'actualisation du répertoire courant du côté client
- Améliorer l'architecture
- Pensez à bien vérifier mes erreurs

VI. Sources

- Mon laboratoire n°3
- Cacher la console :
    - https://stackoverflow.com/questions/18260508/c-how-do-i-hide-a-console-window-on-startup
- Exécution et récupération de la commande système
    - https://stackoverflow.com/questions/34074112/when-to-call-pclose
    - https://stackoverflow.com/questions/38876218/execute-a-command-and-get-output-popen-and-pclose-undefined
    - https://stackoverflow.com/questions/34074112/when-to-call-pclose
- Liens pour les sockets
    - https://stackoverflow.com/questions/5971332/redefinition-errors-in-winsock2-h
    - https://learn.microsoft.com/en-us/windows/win32/api/ws2tcpip/
    - https://learn.microsoft.com/en-us/windows/win32/api/winsock2/
    - https://learn.microsoft.com/en-us/windows/win32/api/ws2tcpip/nf-ws2tcpip-inet_pton
    - https://learn.microsoft.com/en-us/windows/win32/api/ws2def/ns-ws2def-sockaddr_in
- Liens sur les sockets
    - https://learn.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock
    - https://stackoverflow.com/questions/5971332/redefinition-errors-in-winsock2-h
    - https://www.geeksforgeeks.org/socket-programming-cc/
    - https://stackoverflow.com/questions/8480640/how-to-throw-a-c-exception
- Liens sur l'ouverture, l'écriture de fichiers
    - https://en.cppreference.com/w/cpp/filesystem/create_directory
    - https://en.cppreference.com/w/cpp/iterator/istreambuf_iterator
    - https://cplusplus.com/doc/tutorial/files/