TZ20 - Compte rendu

PINARD Maxime - LAZARE Lucas

2015

I TSiD

TSiD est un projet de Lazare Lucas et Pinard Maxime. Ce projet consiste en la conception et la programmation d'un serveur qui permettra de gérer des fichier par internet pour pouvoir y accèder de n'importe ou et les partager avec d'autres personnes. Sur ce serveur chaque utilisateur aura un espace privé ou lui seul aura accès pour y stocker des fichers personnels ou autre. Un espace publique sera aussi disponible pour partager des fichier avec les autres membres. Dans l'espace publique un utilisateur aura la possibilité de supprimer un fichier peu de temps apres l'avoir partagé, au cas ou il se serai trompé, mais après ce laps de temps un système de vote par tout les utilisateurs se mettra en place pour la suppression du fichier.

II Définitions utiles

Un Réseau:
Un réseau informatique est un groupe de machines reliées entre elles pour échanger des informations. Les informations peuvent être transmises via câbles (transmission par signal electrique), fibre optique (ondes lumineuses) ou ondes radio, qui forment un support permettant aux appareils de communiquer en utilisant des protocoles (ex: IEEE 802.11: le wifi, 10BASE5...).

Protocole IP:
Un protocole définit la manière dont les informations sont echangées entre les différents équipements du réseau. Des logiciels, dédiés a la gestion de ces protocoles, sont instalés sur les equipements intermédiaires (ex: routeurs...). Le protocole le plus répandu étant l'Internet Protocole (IP), permettant d'acheminer des paquets de données. Pour cela, l'adresse IP de l'expéditeur et du destinataire sont ajouté au paquet de données ainsi que différentes autres données qui permettent de controler l'acheminement des données. Cela forme un paquet IP.

Adresse IP:
L'adresse IP est une adresse attribuée à chaque équipement connecté directement au réseau internet (nottement box du FAI). Celle-ci est composé de 4 (IPv4) ou 6 (IPv6) séquence de nombres de 0 à 255 et est commune à tout les équipements du réseau local. Par exemple, si vous êtes connectés à une box ayant pour adresse 19.0.92.13, vous aurez la même adresse si vous allez sur internet. Afin que votre routeur puisse reconnaitre les différents équipement du réseau local, celui-ci vous assigne une adresse IP Locale, qui n'est valable que pour le réseau local. Comme l'adresse IP "Internet", celle-ci est unique sur le réseau où vous êtes.

Routage IP:
Le paquet IP est ensuite déposé sur l'ordinateur le plus proche, souvent le serveur du FAI (Fournisser d'Acces Internet) qui va ensuite le rediriger sur un autre ordinateur/serveur... jusqu'à arriver a celui du FAI du destinataire qui va ensuite transmettre le paquet au destinataire.

Port:
Une machine peut se connecter à une autre machine en utilisant son adresse IP. Cependant, une machine faisant serveur web fourni parfois plusieurs services (site web HTTP, serveur FTP, messagerie IMAP, accès SSH, ...). Dès lors, comment indiquer au serveur quel système nous souhaitons utiliser ? C'est là qu'intervient le numéro de port, envoyé par la machine qui se

connecte : 20 pour le FTP, 22 pour le SSH/SFTP, 80 pour l'HTTP, 143 pour l'IMAP,... Le port est en fait un numéro logique au niveau du serveur, qui lui permet de savoir quoi faire avec le client. Un port peut être dans deux états : ouvert ou fermé. Si celui-ci est fermé, la machine n'écoute pas ce qui y arrive, c'est à dire que les requêtes effectués en passant par celui-ci sont ignorées.

Socket:
Le socket, quant à lui, est tout simplement l'association entre le programme en cours, le port local (du client) et l'ip locale, avec l'ip du serveur, son numéro de port. Y est enfin indiqué le port local ouvert, afin que le serveur puisse répondre au client.

La Programmation Orientée Objet:
La POO est un des grands paradigmes de la programmation, c'est à dire une des grandes façon d'écrire un programme. Les principaux autres paradigmes sont la programmation impérative et la programmation déclarative. Comme son nom l'indique, c'est un type de programmation qui utilise les objets pour arriver à ses fins. Mais alors, qu'est-ce qu'un objet ? C'est une sorte de boite contenant des variables (dans le cas d'un socket, il peut s'agir par exemple de l'adresse de sortie, du port, ...), et des fonctions. Les variables sont habituellement inaccessibles depuis l'exterieur de l'objet, contrairement aux fonctions qu'il contient. On peut se le représenter comme étant une boite noire dont on ne voit pas le contenu. On peut actionner des interrupteurs qui modifieront le contenu de la boite et ce qui en ressort, mais on ne sait rien des détails de ce qu'il se passe. C'est en quelque sorte une sécurité : le programmeur ne peut pas faire n'importe quoi avec l'objet, à moins qu'il n'essaye vraiment d'aller dans ce sens, car on est obligé de passer par ses fonctions pour le modifier.

Application Programming Interface:
Une API ou "interface de programmation" est un ensemble de classes, méthodes et fonctions fournies par un logiciel pour la création d'autre logiciels.

III Ce que nous utiliseront

Language de programmation
Nous utiliseron le language C++ qui est une évolution du C, en effet il apporte de nouvelles fonctionalités dont les principales sont la POO et la surcharge d'opérateurs et de fonctions.

API
Nous utiliseron l'API SFML qui nous permettra de gérer plus facilement le réseau.

- La classe `sf::Socket (#include <Socket.hpp>)`
  - La base pour les autres sockets
- La Classe `sf::TcpSocket (#include <TcpSocket.hpp>)`
  - Dépend de `sf::Socket`
  - `TcpSocket()` : créé un Socket

  - `Status connect(const IpAddress &remoteAddress, unsigned short remotePort , Time timeout=Time::Zero)` : permet au client de lancer une connection. Renvoie le status du socket
  - `void disconnect()` : déconnecte le client du server
  - `Status send(Packet &packet)` : envoi un packet au client/serveur préalablement connecté. Renvoie le status du socket.
  - `Status receive(Packet &packet)` : récupère un packet provenant d'un client/serveur préablement connecté. Renvoie le status du socket.
  - `void setBlocking(bool blocking)` : permet de mettre un socket en position bloquante/non bloquante.

- La Classe `sf::Packet (#include <Packet.hpp>)`
  - `Packet()` : créé un packet
  - `append(const void* data, std::size_t sizeInBytes)` : ajoute des données au packet.
  - `const void* getData()const` : renvoie un pointeur vers les données contenues dans le packet.
  - `std::size_t getDataSize()const` : renvoie le nombre de bytes contenues dans le packet.
  - `bool operator BoolType()const` : testes la validité du packet.
  - `Packet& operator<< (bool data)` : surchage de <<
  - `Packet& operator>> (bool& data)` : surcharge de >>

- La Classe `sf::TcpListener (#include <TcpListener.hpp>)`
  - un socket qui permet d'attendre et d'accepter un nouveau client
  - `TcpListener ()` : créé un listener
  - `Status listen (unsigned short port)` : commence a écouter pour la connection d'un client sur le port donné, retourne le status du socket
  - `void close ()` : arrete d'écouter pour la connection d'un client
  - `Status accept (TcpSocket &socket)` : accepte un client et retourne le status du socket

- La Classe `sf::SocketSelector (#include <SocketSelector.hpp>)`
  - un multiplexer (selector) qui permet de lire plusieurs sockets
  - `SocketSelector ()` : créé un selector
  - `void add (Socket &socket)` : ajoute un socket au selector
  - `void remove (Socket &socket)` : supprime un socket du selector
  - `bool wait (Time timeout=Time::Zero)` : attend jusqu'a ce que un ou plusieurs socket soient pret
  - `bool isReady (Socket &socket)const` : teste un socket pour savoir si il est pret

- La Classe `std::thread (#include <thread>)`
  - multithreading standard
  - `thread(public member function)` : créé un thread
  - `void join()` : mets la fonction en pause jusqu'à ce que le thread se termi
  - `void detach()` : délie le thread de la fonction l'ayant appelé. En particulier, permet de quitter la fonction appelante même si le thread n'est pas terminé. Après avoir appelé cette fonction, le thread ne peux plus joindre et la variable peut être détruite sans problème.

IV Les fonctionalités

Programmes prévus

- Serveur pour Linux
- Client pour Linux
- Client pour Windows

Coté serveur

- Accès au serveur uniquement aux possesseurs de compte
- Un espace "public", où les membres déposent des fichiers à partager (Rangés par dossiers et catégories)
- Un espace "privé", auquel seul le membre ayant déposés ses fichiers peut avoir accès (entre 2 et 5 Gio)
- Pour les fichiers de l'espace public, le pseudo du posteur est visible
- Un posteur peut supprimer son propre fichier de l'espace public sous 24h
- Un log des connections et tentatives de connections avec IP (un fichier log par mois) (Nom d'utilisateur, réussite ou non de la connection, ip fournie, date, ...)
- Empêcher le multi-compte
  - Logiciel client qui stocke une id et son mot de passe dans un dossier système (administration nécessaire une seule fois) $->$ Un executable pour windows et un pour linux (pas le même dossier d'enregistrement de l'id/mdp)
  - Ces données seront enregistré dans un fichier nommé [nom].dll
  - Le mot de passe est crypté. $->$ Code César avec pour clef un Hash du pseudo de l'utilisateur
  - Celui-ci doit être modifiable.
- Le serveur ne stocke pas les mots de passes, seulement leur hash

- Charte à l'inscription >> droit d'utilisation des cookies, pas de fichier illégal sur le serveur,...
- On n'autorise qu'une seule fois un pseudo et une adresse mail
- L'identifiant ne sera pas le pseudo, mais l'adresse mail
- Les pseudos des administrateurs sont précédés d'un @ ou d'un %
- L'uploader peut inclure une description du fichier (250 caractères)
- Lorsqu'un membre veut inviter quelqu'un, il doit entrer son adresse mail. On envoi à cette adresse un mot de passe généré aléatoirement. Connection sous 48h obligatoire. Sinon, compte supprimé. Le pseudo est alors à choisir.
- Timeout des sessions (temps à définir)
- Les utilisateurs peuvent faire des sous dossiers publics
- Possibilité d'envoyer un mail à un membre via le serveur (sans connaitre son adresse mail); l'utilisateur peut choisir de désactiver ce service.

Coté client

Pas de GUI

Commandes:

gnuk get new user key – generate a key for a new user to register

cd path Change remote directory to "path"

help Display this help

? Synonym for help

put local-path [remote-path] Upload a file

get remote-path [local-path] Download a file

pwd Display remote working directory

exit Quit sftp

quit Quit sftp

bye Quit sftp

rename oldpath newpath Rename remote file

rmdir path Remove remote directory

rmdir -f path Remove remote directory even if it is not empty

rm path Delete remote file

version Show client version

passwd Change Password

tma path Display Readme

df Display memory usage

deleteaccount Delete the account and its personal datas. Asks for passwd. Also removes its pseudo from the list

msg "text" Sends a message to admins

ls List file in the actual directory

V Problèmes rencontrés

SFML sur RaspBerry pi
SFML n'étant pas disponible pour les processeurs ARM (présent sur le RaspBerry pi) il sera nécéssaire de compiler nous-même SFML avant de l'installer.

L'envoi de données
Lors de nos premiers essais d'envoi de fichiers sous forme de tableaux de données nous avons rencontrés un problème: les packets reçus n'étaient pas de meme taille que les packets reçus nous allons donc utiliser la classe `sf::Packet (#include <Packet.hpp>)` fournie par SFML pour remédier a ce problème. En effet cette classe assure qu'a l'arrivé les paquets sont identiques et dans le même ordre que les originaux.