Dossier de gestion de projet

CPP-SPIDER

Ce document a pour but d'expliquer 3 grands axes :

1- **Le protocole de communication**

2- **Les logs et leurs formatages**

3- **Les classes du projet**

Table des matières

Les codes _____ 1
Les logs et leurs formatages _____ 4
Les classes _____ 6

Le protocole de communication est présent afin de pouvoir simplifier la communication entre le serveur et le client pour chaque groupe de travail.

Les codes

Les codes spécifiques au protocole

Le protocole de communication possède 7 codes bien identifiables.
Voici les codes et leurs explications :

0000 : Déconnection d'un client du serveur
0001 : Connection d'un client au serveur
0002 : Aucune erreur sur la ligne reçue par le serveur
0003 : Erreur sur la ligne reçue par le serveur
0004 : Préparation de l'envoi de fichier
0005 : Fin du fichier envoyé
0006 : Envoie de la clé de chiffrement SSL

Ces codes sont les fondamentaux du protocole de communication.

Le code 0000 (Déconnection d'un client)

Le code 0000 se traduira par la déconnection d'un client.
Par exemple :

- ➔ (Serveur) 0000 : Client [ID] [IP] is deconnected

Le code 0001 (Connection d'un client)

Le code 0001 se traduira par la connexion d'un nouveau client.
Par exemple :

- ➔ (Serveur) 0001 : New client connected [ID] [IP]

Le code 0002 (Aucune erreur)

Le code 0002 se traduira par une réponse positive du serveur sur la ligne reçue.
Par exemple :

- ➔ Client_connected
- ➔ 0002 : 10 clients connected to the server

Le code 0003 (Erreur)

Le code 0003 se traduira par une erreur ou une commande non reconnue par le serveur.
Par exemple :

- ➔ (Client) Hello
- ➔ (Serveur) 0003 : Error command [HELLO]

Le code 0004 (Préparation à l'envoi de fichier)

Le code 0004 se traduira par une demande d'envoi de fichier.
Par exemple :

- ➔ (Client) 0004 : SendFile
- ➔ (Serveur) 0002 : Command Ok

Le code 0005 (Envoi et Fin de fichier)

Le code 0003 se traduira par une erreur ou une commande non reconnue par le serveur.
Par exemple :

- ➔ (Client) File
- ➔ (Client) 0005 : End of File
- ➔ (Serveur) 0002 : Command OK || 0003 : Error Command

Si le serveur renvoie 0003 (Code d'erreur), le serveur repasse donc en attente de fichier.

Le code 0006 (Envoie de la clé ssl)

Le code 0006 se traduira par l'envoi de la clé ssl pour le chiffrement des données.
Par exemple :

- ➔ (Serveur) 0006 : Here's some good stuff
- ➔ (Serveur) Encryptions key
- ➔ (Client) 0002 : Command Ok || 0003 : Error

Les codes

Si le serveur renvoie 0003 il mettra fin à la connexion avec le client. Sinon il passera en attente d’envoi de fichier, donc en code 0004

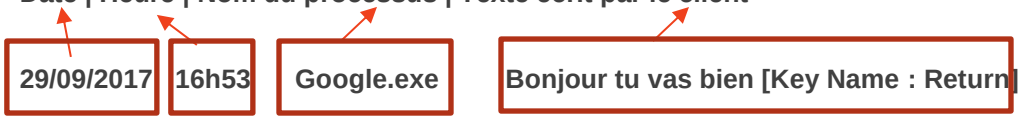
Les logs et leurs formatages

1. Introduction

Les logs seront associés au client, ici tout deux des classes (Voir UML du serveur).

2. Le formatage des logs

Date | Heure | Nom du processus | Texte écrit par le client



3. Les touches claviers simples et spéciales

Certaines touches du clavier seront formatées spécialement comme ci-dessus (pour la liste précises Les touches spéciales).
Le formatage : **[Key Name : Le nom de la touche]**.
Pour les autres touches, il n'y a pas de formatage particulier, elles seront écrites ainsi : « je vais bien »

4. La souris

Pour la souris le formatage diffère légèrement :
[Mouse : Action : <x1, y1> ; (x2, y2)].

<x1, y2> : Sont des paramètres persistants.

(x2, y2) : Sont des paramètres qui sont facultatifs pour certaines actions.

Les logs et leurs formatages

Action : L'action effectuée sur la souris (Clic, Mouvement).
Comme ci-dessus, nous pouvons apercevoir qu'il existe deux actions, clic et mouvement.

L'action **Clic** présente 3 type de clic différent, qui sont le **clic gauche**, **droit** et pour finir le **clic molette**. Ils seront symbolisés comme ceci : [Mouse : Click : L (1) : x, y]. Ici le x et y représente la position au moment de l'action clic.

L'action **Mouvement**, sera formatée de la même manière que l'action **clic** mais présentera les paramètres supplémentaires dis facultatif. Ainsi le formatage sera donc :
[Mouse : Move : x1, y1 ; x2, y2].
Respectivement **x1** et **y1** sont les positions de départ lors du mouvement de la souris, et **x2**, **y2** sont les positions de fin de la souris.

Les classes

Introduction

Ce projet est séparé en deux parties bien distincte. L'une est la partie serveur, l'autre la partie client. Le descriptif des classes sera alors séparé en deux parties.

Les classes du serveur

- AServer
- AEncryption
- ClientTcpConnection
- DBMongo
- EOpenSSL
- IDBManager

AServer

Cette classe permet l'initialisation du serveur (Initialisation de la connexion, initialisation de la base de données) puis la connexion des clients.

a) Contenu de la classe

1) Méthodes

Nom	Description	Type	Paramètres
AServer()	Constructeur	Boost ::asio ::io_service	réference
		Unsigned short	int
Virtual ~AServer()	Destructeur	N/A	N/A
Void createClient()	Création d'un client	N/A	N/A
Virtual void run()	Lancement de la boucle serveur	N/A	N/A

2) Attributs

Nom	Description	Type
Clients	Conteneur clients	Std ::vector<spider ::CLientTcpConnection ::pointer>
Port	Port de connexion	Unsigned short
Accept	Type de connexion	Tcp ::accept
io_serv	Gestion I/O Boost	Boost ::asio ::io_service
dbManager	Gestion de la base de données	Pointeur IDBManager

AEncryption

Cette classe est une abstraction permettant l'ajout futur d'autres méthodes de chiffrement.

a) Contenu de la classe

1) Méthodes

Nom	Description	Type	Paramètres
AEncryption()	Constructeur	N/A	N/A
Virtual ~AEncryption()	Destructeur	N/A	N/A
Virtual const std ::string init() const	Initialisation du chiffrement	Std ::string const	référence
Virtual const std ::string encrypt() const	Chiffrement du message	Std ::string const	référence
Virtual const std ::string decrypt() const	Déchiffrement du message	Std ::string const	référence

2) **Attributs**

Nom	Description	Type
active	Connexion active ou pas	bool
Ip	IP de connexion	Std ::string

EOpenSSL

Cette classe permet la gestion du chiffrement de la communication via OpenSSL

a) Contenu de la classe

1) **Méthodes**

Nom	Description	Paramètre	Type
EOpenSSL()	Constructeur	N/A	N/A
~EOpenSSL()	Destructeur	N/A	N/A
const std ::string init() const	Initialisation de l'encryptage	N/A	N/A
const std ::string encrypt() const	Encryptage	N/A	N/A
Virtual const std ::string decrypt() const	Décryptage	N/A	N/A
bool generate_keys()	Génération de clé	N/A	N/A
bool initRsaFromFile()	Initialisation de la clé grâce a un fichier	N/A	N/A
const std::string handShake()	Mélange	N/A	N/A

2) **Attributs**

Nom	Description	Type
publicKey	Clé public	Pointeur RSA
privateKey	Clé privé	Pointeur RSA

ClientTcpConnection

Cette classe permet l'ajout d'un client et de toute ses actions.

a) Contenu de la classe

1) **Méthodes**

Nom	Description	Paramètre	Type
ClientTcpConnection()	Constructeur	boost ::asio ::io_service	Référence
		AEncryption	Pointeur
Virtual ~ ClientTcpConnection ()	Destructeur	N/A	N/A
static pointer create()	Création d'un client	Boost ::asio ::io_service	Référence
		AEncryption	Pointeur
void setIp()	Etablir adresse IP	Std ::string const	N/A
Void init()	Initialisation du client	N/A	N/A
boost ::system ::error_code sendMessage()	Envoi des messages d'erreurs	Std ::string	N/A
void handleWrite()	Gestion de l'écriture des messages	Const boost ::system ::error_code	N/A

void handleRead()	Gestion de la lecture des messages	Const boost ::system ::error_code	N/A
Void listenRead()	Ecoute de la réception des messages	N/A	N/A
Void closeConnection()	Déconnection du client	N/A	N/A
bool handleConnexionState();	Vérification de l'état de la connexion	N/A	N/A
bool handleWaitingFileState();	Vérification de l'état de la connexion	N/A	N/A
bool handleRcvFileState();	Vérification de l'état de la connexion	N/A	N/A

2) **Attributs**

Nom	Description	Type
_state	Etat de la connexion	State
ip	IP de connexion	Std ::string
socket;	Socket permettant la connexion	tcp::socket
m_mess	Chaine de caractère comportant le message	std::string
buf	Buffer de message	Boost ::array
rsa	Clé rsa pour l'encodage	AEncryption *
request	Requête	std::map

Communication

Cette classe permet la gestion de la communication

a) Contenu de la classe

1) Méthodes

Nom	Description	Paramètre	Type
Communication()	Constructeur	N/A	N/A
virtual ~Communication()	Destructeur	N/A	N/A
void parseBuffer()	Parsing de la ligne reçue	N/A	N/A
void addData(std::string read)	Ajout de données	Std::string	N/A

2) Attributs

Nom	Description	Type
log	Log d'un client	Std::fstream
fileName	Nom du fichier	Std::string
Buffer	Buffer	Std::string

DBMongo

Cette classe permet la gestion de la base de données mongo

a) Contenu de la classe

1) Méthodes

Nom	Description	Paramètre	Type
DBMongo()	Constructeur	N/A	N/A
~DBMongo()	Destructeur	N/A	N/A
bool insert(const std::string & insert)	Insertion dans une base de données	Const std::string	Référence
bool init()	Vérification de l'initialisation	N/A	N/A

IDBManager

Cette est une classe virtuelle permettant la gestion de plusieurs bases de données différentes

b) Contenu de la classe

3) Méthodes

Nom	Description	Paramètre	Type
IDBManager()	Constructeur	N/A	N/A
virtual ~IDBManager()	Destructeur	N/A	N/A

virtual bool insert(const std::string & insert)	Insertion en base de données	N/A	N/A
virtual bool init()	Initialisation	N/A	N/A

Les classes du client

- IRecorder
- Keyboard
- Mouse
- IClient
- LinuxClient
- AEncryption
- EOpenSSL
- Communication
- IWorm
- Keylogger
- Process
- TaskManager

IRecorder

Cette interface permet un ajout de périphériques supplémentaires.

a) Contenu de la classe

1) Méthodes

Nom	Description	Paramètres	Type
Virtual ~IRecorder()	Destructeur	N/A	N/A
Virtual void write()	Ecriture dans le fichier de log	Const std ::string	Référence
Virtual void read()	Lecture du fichier de log	N/A	N/A

Keyboard

Cette classe s'occupe de l'enregistrement des I/O venant d'un clavier.

a) Contenu de la classe

1) Méthodes

Nom	Description	Paramètres	Type
Keyboard()	Constructeur	N/A	N/A
Virtual ~Keyboard()	Destructeur	N/A	N/A
Virtual void write()	Ecriture dans le fichier de log	Const std ::string	Référence
Virtual void read()	Lecture du fichier de log	N/A	N/A
void getMajState()	Vérification des majuscules	Unsigned char	Pointeur
void checkCapital()	Verification des touches spécifiques	Unsigned int	N/A
std::string readKey()	Lire touches appuyées	N/A	N/A
std::string manageKeys()	Manager des I/O	Unsigned int	N/A
void allUnlock()	Déverouiller les touches de clavier	N/A	N/A
void shiftLock()	Touche shift appuyée	N/A	N/A
void capitalLock()	Touche MAL appuyée	N/A	N/A

2) Attributs

Nom	Description	Type
lockOn	Tableau de booléen pour savoir si certaines touches spécifiques sont appuyées	Bool[2]
capital	Touche MAJ appuyée	bool
record	Enregistrement des inputs	Std ::string

Mouse

Cette classe s'occupe de l'enregistrement des I/O venant d'une souris.

a) Contenu de la classe

1) Méthodes

Nom	Description	Paramètres	Type
Mouse()	Constructeur	N/A	N/A
Virtual ~Mouse()	Destructeur	N/A	N/A
Virtual void write()	Ecriture dans le fichier de log	Const std ::string	Référence
Virtual void read()	Lecture du fichier de log	N/A	N/A
std::string getCursorPos()	Position du curseur	N/A	N/A
std::string manageMoves()	Manager des mouvements de souris	N/A	N/A
std::string manageClicks	Manager des clics de souris	N/A	N/A
std::string readMouse()	Lecture des mouvements/clics de souris	N/A	N/A

2) Attributs

Nom	Description	Type
clickNames	Vecteur des noms de clics	Std ::vector<std ::string>
lastPos	Dernière position du curseur enregistré	Int[2]
record	Enregistrement des inputs	Std ::string
start	Début de l'enregistrement	Time_t

IClient

Cette interface permet l'ajout de systèmes d'exploitation supplémentaires pour le client.

a) Contenu de la classe

1) Méthodes

Nom	Description	Paramètres	Type
Virtual ~IClient()	Destructeur	N/A	N/A
Virtual bool run()	Lancement du client	N/A	N/A

LinuxClient

Cette classe gère le fonctionnement du client sous un système UNIX.

a) *Contenu de la classe*

1) Méthodes

Nom	Description	Paramètres	Type
LinuxClient()	Constructeur	N/A	N/A
Virtual ~LinuxClient ()	Destructeur	N/A	N/A
Virtual bool run()	Lancement du client	N/A	N/A

2) Attributs

Nom	Description	Type
taskManager	Object permettant la gestion des tâches	TaskManager
network	Object qui initialise du réseau et permet de communiquer	Communication
keylogger	Object gérant l'enregistrement des périphériques	Keylogger

AEncryption

Classe abstraite permettant l'ajout futur d'autres méthodes de chiffrement.

a) Contenu de la classe

1) Méthodes

Nom	Description	Type	Paramètres
AEncryption()	Constructeur	N/A	N/A
Virtual ~AEncryption()	Destructeur	N/A	N/A
Virtual const std ::string init() const	Initialisation du chiffrement	Std ::string const	référence
Virtual const std ::string encrypt() const	Chiffrement du message	Std ::string const	référence
Virtual const std ::string decrypt() const	Déchiffrement du message	Std ::string const	référence

2) Attributs

Nom	Description	Type
active	Connexion active ou pas	bool
Ip	Adresse IP de connexion	Std ::string

EOpenSSL

Classe s'occupant de la gestion du chiffrement de la communication client/serveur en OpenSSL.

a) *Contenu de la classe*

1) Méthodes

Nom	Description	Type	Paramètres
EOpenSSL()	Constructeur	N/A	N/A
Virtual ~ EOpenSSL()	Destructeur	N/A	N/A
Void init()	Initialisation du chiffrement	Std ::string const	référence
bool initRsaFromText()	Initialisation de la clé de chiffrement	Std ::string const	référence
Virtual const std ::string encrypt() const	Chiffrement du message	Std ::string const	référence
Virtual const std ::string decrypt() const	Déchiffrement du message	Std ::string const	référence

2) Attributs

Nom	Description	Type
publicKey	Clé public	RSA

Communication

Cette classe gère la communication client/serveur.

a) *Contenu de la classe*

1) Méthodes

Nom	Description	Paramètres	Type
Communication()	Constructeur	N/A	N/A
Virtual ~ Communication()	Destructeur	N/A	N/A
void run()	Lancement de la communication	N/A	N/A

Process

Cette classe s'occupe de la gestion des processus.

a) *Contenu de la classe*

1) Méthodes

Nom	Description	Paramètres	Type
Process()	Constructeur	N/A	N/A
Virtual ~Process()	Destructeur	N/A	N/A
std ::string getActiveWindowTitle()	Obtenir le nom de la fenêtre active	N/A	N/A
std ::string getProcessPath()	Obtenir le chemin d'un processus	Unsigned int	N/A
unsigned int getProcessIdByProcessName()	Obtenir l'ID d'un processus avec son nom	Unsigned int	N/A

unsigned int getProcessIdByWindowName()	Obtenir l'ID d'un process avec la fenêtre active	Const std::string	référence
void hideWindow()	Cacher la fenêtre	N/A	N/A
bool isModified()	Savoir si le processus a été modifié	N/A	N/A

2) Attributs

Nom	Description	Type
lastProcId	L'ID du dernier processus	Unsigned int

TaskManager

Cette classe s'occupe de la gestion des DLL.

a) Contenu de la classe

1) Méthodes

Nom	Description	Paramètres	Type
TaskManager()	Constructeur	N/A	N/A
Virtual ~TaskManager()	Destructeur	N/A	N/A
const std ::string getKeyLogDllPath()	Obtenir le chemin du .dll du keylogger	N/A	N/A
Process & getProcess() ;	Obtenir l'objet Process	Unsigned int	N/A
Bool isInjected()	Obtenir le booléen injected	N/A	N/A

bool injectDll()	Injection du .dll	Const std::string	Référence
		Unsigned int	N/A
bool hide()	Cacher le processus	Const std::string	Référence
Void run()	Lancer le manager de tâches	N/A	N/A

2) **Attributs**

Nom	Description	Type
process	L'objet gérant les processus	Process
KeyLogDllPath	Chemin du .dll	Std ::string
Injected	Savoir le .dll est bien injecté	bool