Manuel à l’usage des développeurs

Projet 2ARC Supinfo 2014 – 2015

Sommaire

[Configuration préalable 2](#_Toc422078327)

[Programmes d’exemple 2](#_Toc422078328)

[Fonctions 3](#_Toc422078329)

[Fonctions ARP 3](#_Toc422078330)

[sendARPRequest 3](#_Toc422078331)

[Fonctions ICMP 4](#_Toc422078332)

[sendICMPRequest 4](#_Toc422078333)

[sendICMPRequests 4](#_Toc422078334)

[Fonctions de transfert de données 5](#_Toc422078335)

[sendFile 5](#_Toc422078336)

[receiveFile 6](#_Toc422078337)

[receiveInt 6](#_Toc422078338)

[sendInt 7](#_Toc422078339)

# Configuration préalable

Avant de pouvoir commencer à utiliser notre bibliothèque, il faudra vous assurer de plusieurs choses.

Premièrement, les fichiers « 2arc\_dll\_release.dll », « 2arc\_dll\_release.lib » et « dll.h » devront être placés avec le code source de votre projet. N’oubliez pas d’ajouter « dll.h » au projet de votre IDE.

Deuxièmement, il va falloir les intégrer à votre projet. Avec Visual Studio 2013, il faudra rajouter les lignes suivantes à votre code :

#include "dll.h"

#pragma comment(lib, "2arc\_dll\_release")

#pragma comment(lib, "iphlpapi.lib")

#pragma comment(lib, "ws2\_32.lib")

Ce qui inclura la bibliothèque à votre programme.

Avec Codeblocks, il faudra uniquement ajouter #include "dll.h" à votre code, puis aller dans « Project > Build Options > Linker Settings » et ajouter iphlpapi.a, ws2\_32.a ainsi que notre bibliothèque.

Après ces manipulations, vous devriez être en mesure d’utiliser la bibliothèque.

Pour utiliser nos fonctions sans problème, veillez à ce que les ports 8888 et 20000 soient disponibles pour votre projet, et que rien ne bloque le trafic sur ces ports (Pare-feu, OS).

Concernant la version released de tout projet utilisant notre librairie, il faudra inclure « 2arc\_dll\_release.dll » et « 2arc\_dll\_release.lib » au dossier contenant l’exécutable final, ce sans quoi le programme refusera probablement de se lancer.

# Programmes d’exemple

Quatre programmes sont livrés avec cette bibliothèque. Ils illustrent des utilisations possibles des fonctions ARP, ICMP et de transfert de données (un client et un serveur).

Ils ont été testés sur des machines virtuelles Windows 8 Pro.

En effet, mis à part le programme destiné à tester les fonctions ICMP, les programmes utilisent l’API Windows pour afficher une interface graphique plus agréable qu’une console.

Ils permettent donc de récupérer une adresse MAC à partir d’une adresse IP, de pinger une adresse IP et de transférer tout type de fichier ou nombre d’un serveur vers un client.

# Fonctions

## Fonctions ARP

### string sendARPRequest(char \*ipAddress)

Cette fonction permet de récupérer l’adresse MAC d’un appareil à partir de son adresse IP, via une requête ARP.

L’appareil cible n’a pas besoin de s’attendre à recevoir la requête.

Argument entrant

ipAddress : l’adresse IP à laquelle on souhaite envoyer à la requête. Doit être formatée comme suit : « X.X.X.X » avec X un nombre compris entre 0 et 255.

Argument sortant

L’adresse MAC retournée en string au format « XX:XX:XX:XX:XX:XX » où X est un caractère alphanumérique.

Retourne "No answer received." si rien n’a été retourné. Typiquement, l’appareil ciblé est hors-tension ou l’adresse IP fournie n’est pas correcte.

## Fonctions ICMP

### bool sendICMPRequest(char \*ipAddress, PULONG hopCount, PULONG rtTime)

Cette fonction envoie un ping à l’adresse IP spécifiée et retourne diverses statistiques.

L’appareil cible n’a pas besoin de s’attendre à recevoir la requête.

Arguments entrants

ipAddress : l’adresse IP à laquelle on souhaite envoyer à la requête. Doit être formatée comme suit : « X.X.X.X » avec X un nombre compris entre 0 et 255.

hopCount : un pointeur sur un ULONG. Destiné à contenir le nombre de sauts nécessaires pour atteindre la cible du ping.

rtTime : un pointeur sur un ULONG. Destiné à contenir le temps en millisecondes nécessaire pour atteindre la cible du ping.

Argument sortant

Retourne TRUE si le ping a été retourné par l’appareil cible, retourne FALSE sinon.

Un autre moyen de vérifier que l’appareil cible ait bien répondu est de contrôler la valeur de hopCount et rtTime. S’ils sont passés à la fonction avec une valeur négative et que le ping échoue, les valeurs resteront négatives.

### bool sendICMPRequests(char \*ipAddress, PULONG hopCount, PULONG rtTime)

Une modification de la fonction sendICMPRequest. Cette fonction envoie 10 000 pings à un intervalle d’une seconde et retourne FALSE si l’un échoue.

Se référer à sendICMPRequest pour en savoir plus sur son fonctionnement.

## Fonctions de transfert de données

### bool sendFile(const char \*fileName)

Cette fonction permet d’envoyer un fichier par le réseau.

Une fois cette fonction lancée, la machine va attendre une connexion de l’appareil qui doit recevoir le fichier. Il s’agit d’une fonction bloquante, ce qui signifie que le programme attendra une connexion avant toute action. Veillez à créer un thread pour cette fonction si vous ne souhaitez pas que votre programme se bloque.

La connexion est acceptée dès qu’une machine tente de s’y connecter.

Une fois la connexion établie, le fichier est envoyé, en plusieurs paquets si le fichier est trop gros pour être envoyé en une seule fois.

Le client devra créer un fichier de la même extension que celui qui est envoyé. Par exemple, on peut envoyer « test.txt » et mettre le contenu de test.txt dans « fichier\_recu.txt », mais pas dans « fichier\_recu.jpg ».

Arguments entrants

fileName : le nom du fichier à envoyer, avec son extension. Par exemple, « test.txt » ou « image.png ». Il peut aussi s’agir d’un chemin relatif ou absolu.

Argument sortant

Retourne TRUE si le transfert a été effectué avec succès.

Retourne FALSE sinon.

### bool receiveFile(char \*ipAddress, const char \*fileName)

Cette fonction permet de recevoir un fichier de l’adresse IP spécifiée.

Lorsque cette fonction est utilisée, une connexion est établie entre cette machine et l’appareil cible (qui envoie le fichier). Il est nécessaire que l’appareil cible ait déjà lancé avec succès la fonction sendFile et, de ce fait, attende une connexion.

L’extension du fichier à créer doit être la même que celle du fichier envoyé.

Arguments entrants

ipAddress : l’adresse IP à laquelle on souhaite se connecter. Doit être formatée comme suit : « X.X.X.X » avec X un nombre compris entre 0 et 255.

fileName : le nom du fichier à créer, avec son extension. Par exemple, « test.txt » ou « image.png ». Il peut aussi s’agir d’un chemin relatif ou absolu.

Argument sortant

Retourne TRUE si le transfert a été effectué et que le fichier est créé avec succès.

Retourne FALSE sinon.

Si un fichier du même nom existe dans le dossier de destination, il sera écrasé et remplacé par le nouveau.

### int receiveInt(char \*ipAddress)

Cette fonction permet de recevoir un nombre (int) de l’adresse IP spécifiée.

Au lancement de la fonction, une connexion est établie entre cette machine et l’appareil cible (qui envoie le nombre). Il est nécessaire que l’appareil cible ait déjà lancé avec succès la fonction sendInt et, de ce fait, attende une connexion.

Argument entrant

ipAddress : l’adresse IP à laquelle on souhaite se connecter. Doit être formatée comme suit : « X.X.X.X » avec X un nombre compris entre 0 et 255.

Argument sortant

Retourne le nombre int reçu si le transfert a été effectué avec succès.

Retourne 0 sinon.

### bool sendInt(char \*ipAddress, int number)

Cette fonction permet d’envoyer un nombre (int) à l’adresse IP spécifiée.

À l’exécution de la fonction, la machine va attendre une connexion de l’appareil qui doit recevoir le nombre. Il s’agit d’une fonction bloquante, ce qui signifie que le programme attendra une connexion avant toute action. Veillez à créer un thread pour cette fonction si vous ne souhaitez pas que votre programme se bloque.

Une fois la connexion établie, le nombre est envoyé.

Cette fonction a été pensée pour pouvoir envoyer des codes entre un client et un serveur, chaque code pouvant correspondre à une requête ou un message d’erreur.

Arguments entrants

ipAddress : l’adresse IP à laquelle on souhaite envoyer le nombre. Doit être formatée comme suit : « X.X.X.X » avec X un nombre compris entre 0 et 255. ipAddress peut aussi être vide "". Dans ce cas, elle acceptera la première connexion qu’elle recevra.

number : le nombre à envoyer.

Argument sortant

Retourne TRUE si le transfert a été effectué et que le fichier créé avec succès.

Retourne FALSE sinon.

train-commander.dev/api/stations

train-commander.dev/api/user/{id}

train-commander.dev/api/user/fb/{fbID}

train-commander.dev/api/user/google/{googleID}

train-commander.dev/api/connect/{email}/{password}

train-commander.dev/api/create/user/{firstname}/{lastname}/{password}/{email}/{newsletter}

train-commander.dev/api/journey(s)/{searchMode}/{startStationID}/{endStationID}/{startTimestamp}

train-commander.dev/api/journey(s)/{searchMode}/{startStationID}/{endStationID}/{minTimestamp}/{maxTimestamp}

train-commander.dev/api/create/history/{cost}/{startStationName}/{arrivalStationName}/{startTimestamp}/{arrivalTimestamp}/{userid}