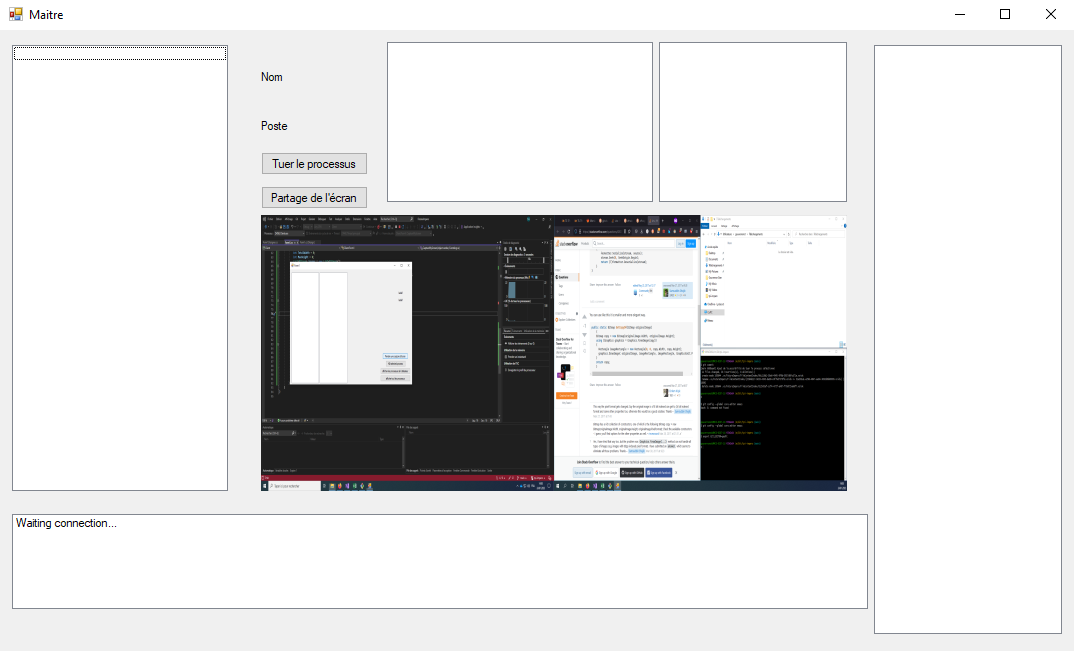
Future Impero



Application de monitoring et d’aide des élèves

Ecole : CIFOM, Ecole technique

Atelier : Salle 207

Langage : C#

Auteur : Stan Gouvernon

Supérieur : Alain Jeanmaire

Date : 24.01.2023-22.03.2023

Table des matières

[Présentation du projet 4](#_Toc129700513)

[Prérequis 4](#_Toc129700514)

[Matériel et logiciels 4](#_Toc129700515)

[Connaissances requises 4](#_Toc129700516)

[Cahier des charges 4](#_Toc129700517)

[But du projet 4](#_Toc129700518)

[Maquettes des applications 5](#_Toc129700519)

[Fonctionnalités prévues 5](#_Toc129700520)

[Client 5](#_Toc129700521)

[Maître 6](#_Toc129700522)

[Communication 6](#_Toc129700523)

[Schéma de fonctionnement 6](#_Toc129700524)

[Schéma de classe 7](#_Toc129700525)

[Gestion du projet 7](#_Toc129700526)

[Mise en place 7](#_Toc129700527)

[Matériel 7](#_Toc129700528)

[Logiciel 7](#_Toc129700529)

[Application cliente 8](#_Toc129700530)

[Interface 8](#_Toc129700531)

[Démarrage 8](#_Toc129700532)

[Trouver les processus lancés par défaut 8](#_Toc129700533)

[Trouver les processus lancés par l’utilisateur 9](#_Toc129700534)

[Arrêter les processus lancés par l’utilisateur 9](#_Toc129700535)

[Capture d’écran des deux moniteurs 9](#_Toc129700536)

[Récupération des urls ouverts 10](#_Toc129700537)

[ClientData 11](#_Toc129700538)

[Recevoir les demandes 11](#_Toc129700539)

[Arrêt 12](#_Toc129700540)

[Classe ClientForm 12](#_Toc129700541)

[Application Maître 13](#_Toc129700542)

[Interface 13](#_Toc129700543)

[Démarrage 14](#_Toc129700544)

[Résumé 14](#_Toc129700545)

[Envoi de requêtes 14](#_Toc129700546)

[Affichage des données client 15](#_Toc129700547)

[Arrêt 15](#_Toc129700548)

[Classe MaitreForm 16](#_Toc129700549)

[Liaisons entre les deux applications 16](#_Toc129700550)

[Connexion 16](#_Toc129700551)

[Socket 17](#_Toc129700552)

[Données 17](#_Toc129700553)

[Images 18](#_Toc129700554)

[Flux d’image 18](#_Toc129700555)

[Demande d’arrêt d’un processus 20](#_Toc129700556)

[Tests 20](#_Toc129700557)

[Méthode de test 20](#_Toc129700558)

[Problèmes et solutions 21](#_Toc129700559)

[Références 21](#_Toc129700560)

[Webographie 21](#_Toc129700561)

[Glossaire 22](#_Toc129700562)

[Bilan 22](#_Toc129700563)

[Améliorations possibles 22](#_Toc129700564)

[Objectifs atteints 22](#_Toc129700565)

[Client 22](#_Toc129700566)

[Maître 23](#_Toc129700567)

[Communication 23](#_Toc129700568)

[Conclusion 23](#_Toc129700569)

[Annexe Websummary 24](#_Toc129700570)

[Annexe Planification 25](#_Toc129700571)

[Annexe Manuel d’utilisateur 26](#_Toc129700572)

[Application Cliente 26](#_Toc129700573)

[Application Maître 26](#_Toc129700574)

[Annexe Journal de travail 26](#_Toc129700575)

# Présentation du projet

## Prérequis

### Matériel et logiciels

2 ordinateurs avec Windows 10.

Les deux ordinateurs sont connectés sur le même réseau local.

Visual Studio 2022 sur l’un des ordinateurs.

### Connaissances requises

Connaissance en développement orienté objet en C#

Connaissance sur l’envoi de données sur le réseau avec les Socket.

## Cahier des charges

La but du projet est de proposer une alternative à l’application Impero, qui permet actuellement à l’utilisateur de l’application maître me monitorer ainsi que d’aider les utilisateurs des applications clientes. Mais Impero doit être installé ce qui ne sera plus possible au moment où les futurs utilisateurs des applications clientes viendront avec leurs propres ordinateurs portables. C’est pourquoi je ne peux pas utiliser d’autres outils qu’il faudrait installer car mon application est portable et doit au maximum reproduire les fonctionnalités d’Impero. Il faut donc que je crée deux applications différentes une pour les utilisateurs des applications clientes et une pour les utilisateurs de l’application maître.

## But du projet

Proposer une alternative à Impero quand il ne pourra plus être installé sur les ordinateurs des futurs utilisateurs des applications clientes. De ce fait il faut donc développer un logiciel portable et le plus léger en ressource possible car il devra tourner sur des ordinateurs portables.

## Maquettes des applications

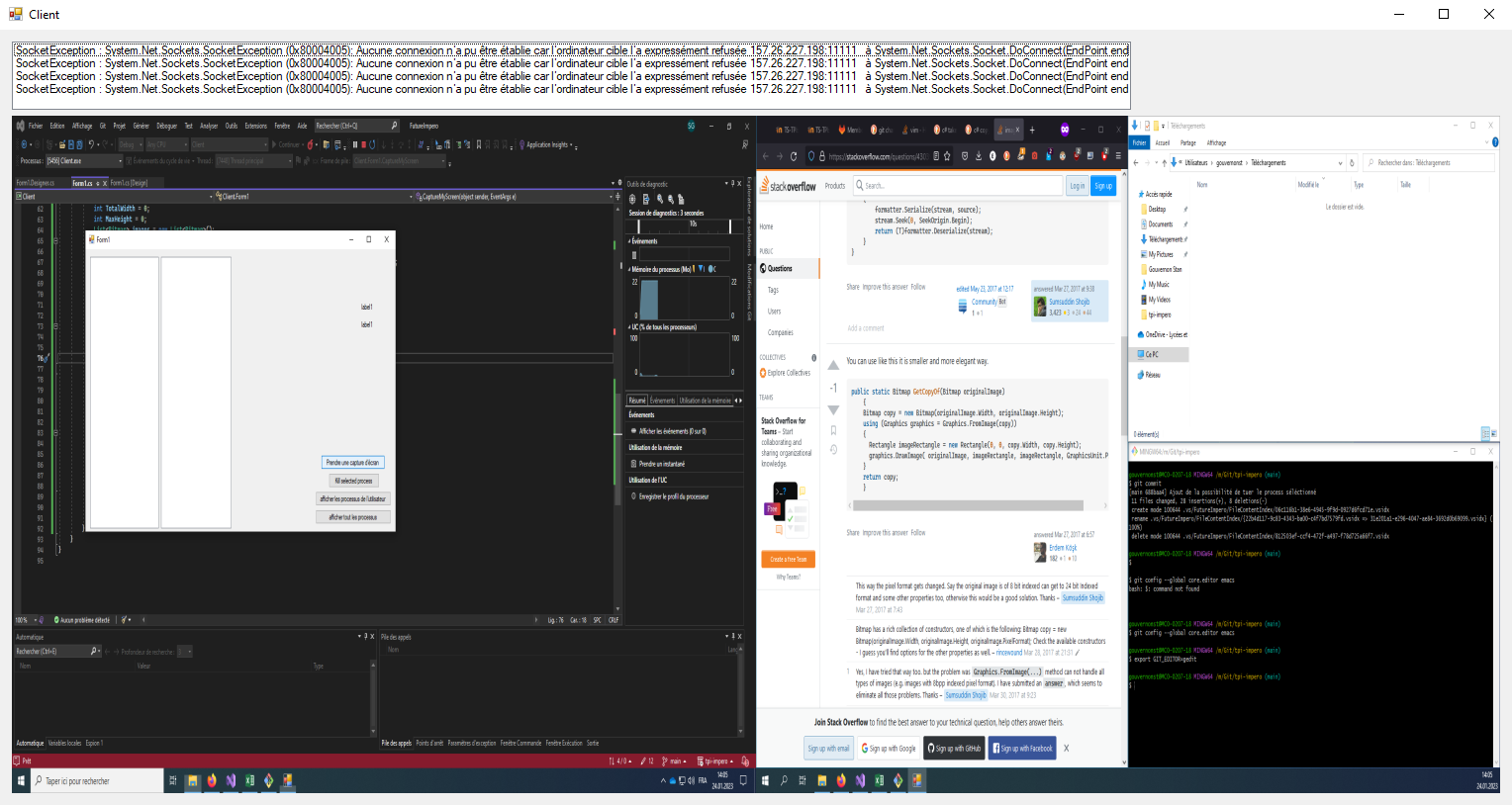


Figure 1:Application cliente en fonctionnement, mode d’emploi page 8

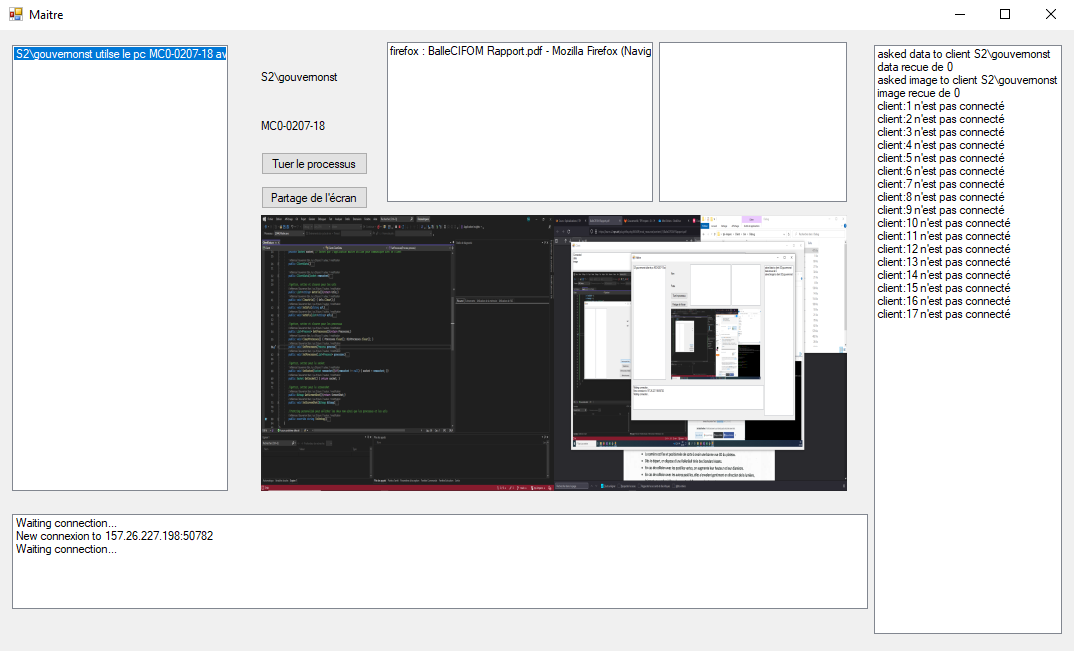


Figure 2.Application maître en fonctionnement, mode d’emploi page 13

## Fonctionnalités prévues

### Client

Prendre une capture d’écran de tous les écrans

Récupération des processus lancé par l’utilisateur.

Récupération de toutes les url ouvertes dans tous les navigateurs

Minimiser en TrayIcon pour cacher l’application.

Afficher l’écran de l’application maître

### Maître

Arrête des processus indésirables des clients

Minimiser en TrayIcon pour cacher l’application.

Afficher les données reçues.

Afficher les captures d’écrans reçues.

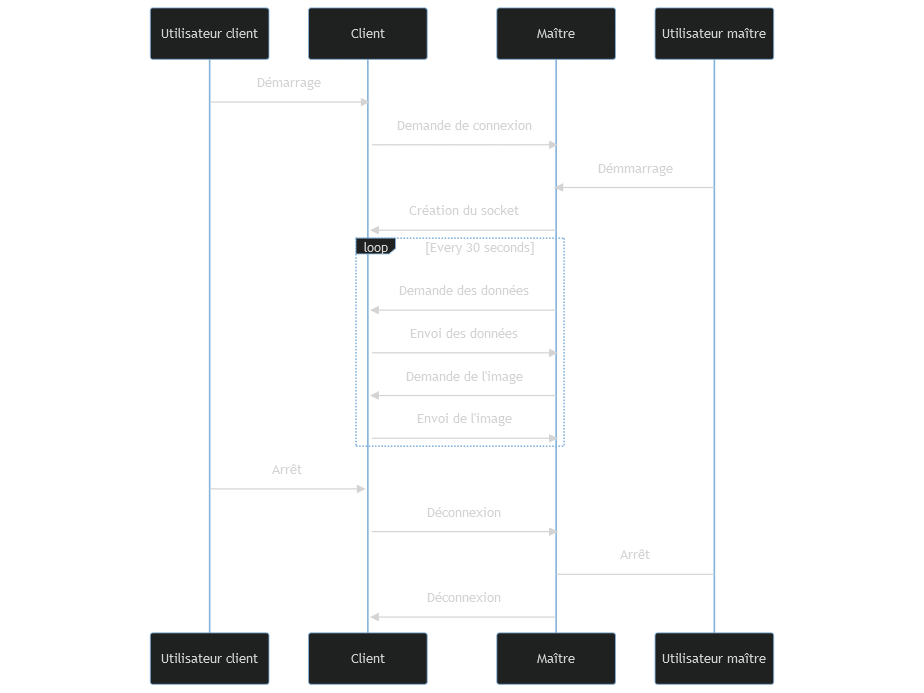
Prendre le contrôle d’un pc client

### Communication

Envoi des données dans des threads séparé.

Transférer les données à haut débit.

## Schéma de fonctionnement



## Schéma de classe



# Gestion du projet

Pour la gestion de projet j’ai utilisé un développement agile, il s’agit d’une méthode de travail faite pour privilégier des rendus réguliers au client et des retours rapides sur ce que l’on a fait. J’ai donc fait plusieurs rendus pour les nouvelles fonctionnalités que j’avais développé.

# Mise en place

## Matériel

L’installation matériel est relativement simple il suffit que deux ordinateurs Windows soient sur le même réseau local. Je ne décrirais pas l’installation de Windows ici. Mais pour la liaison au même réseau local il suffit de connecter les deux ordinateurs au même switch.

## Logiciel

Installation de Visual Studio 2022. Mais elle nécessite tout de même des droits administrateurs. Mais si c’est le cas elle est simple, il vous suffit d’aller sur leur site <https://visualstudio.microsoft.com/fr/> puis de télécharger la version communautaire pour Windows. Á l’installation le seul module qui soit nécessaire est « Développement .NET Desktop »

# Application cliente



## Interface

1 : La listbox lbxConnexion : Permet d’afficher les logs de l’application tel que la connexion au serveur, l’envoi de donné ou l’arrêt d’un processus.

2 : La picturebox pbxMasterScreen : Affiche la dernière capture d’écran effectuée par l’application.

L’interface est simple et ne sera visible aux utilisateurs finaux qu’en cas de partage d’écran de l’application maitre ou si l’utilisateur l’ouvre explicitement pour voir les logs de connexion dans lbxConnexion.

## Démarrage

Au démarrage l’application cliente va essayer de se connecter à une application maître dont on lui a fourni l’IP.

|  |  |
| --- | --- |
| 1 | public ClientForm() |
| 2 | **{** |
| 3 | InitializeComponent**();** |
| 4 | GetDefaultProcess**();** |
| 5 | //lance la connexion à l'application maître |
| 6 | Task**.**Run**(**ConnectToMaster**);** |
| 7 | **}** |

## Trouver les processus lancés par défaut

Pour trouver les processus par défaut il nous suffit au démarrage de l’application de créer une liste de tous les processus lancés à ce moment-là avec Process.GetProcesses().

|  |  |
| --- | --- |
| 1 | public void GetDefaultProcess() |
| 2 | **{** |
| 3 | DefaultProcess**.**Clear**();** |
| 4 | **foreach** **(**Process process **in** Process**.**GetProcesses**().**OrderBy**(**x **=>** x**.**ProcessName**))** |
| 5 | **{**DefaultProcess**.**Add**(**process**.**ProcessName**);}** |
| 6 | **}** |

## Trouver les processus lancés par l’utilisateur

Pour savoir quels processus ont été lancés par l’utilisateur, il suffit de comparer les processus actuels avec les processus par défaut trouvé au point précédent.

|  |  |
| --- | --- |
| 1 | public void GetUserProcess() |
| 2 | **{** |
| 3 | data**.**ClearProcesses**();** |
| 4 | List**<**Process**>** list **=** Process**.**GetProcesses**().**OrderBy**(**x **=>** x**.**ProcessName**).**ToList**();** |
| 5 | **foreach** **(**Process process **in** list**)** |
| 6 | **{** |
| 7 | **if(!**DefaultProcess**.**Contains**(**process**.**ProcessName**))** |
| 8 | **{**data**.**SetProcesses**(**process**);}** |
| 9 | **}** |
| 10 | **}** |

## Arrêter les processus lancés par l’utilisateur

Après avoir récupéré la liste des processus lancé par l’utilisateur il faut pouvoir arrêter le processus indésirable. Pour cela nous envoyons un dictionnaire avec le nom et l’id du processus à l’application maître. Elle peut ensuite quand elle le désire nous renvoyer l’id pour dire à l’application cliente de stopper ce processus.

|  |  |
| --- | --- |
| 1 | public void KillSelectedProcess(int id) |
| 2 | **{** |
| 3 | **try{**Process**.**GetProcessById**(**id**).**Kill**();}** |
| 4 | **catch{**lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**"le processus " **+** id **+** "n'a pas pu être tué"**);** **}));}** |
| 5 | **}** |

## Capture d’écran des deux moniteurs

Prendre une capture d’écran de plusieurs moniteurs n’est pas facile car on ne peut pas capturer plusieurs écrans en même temps. Il faut donc faire plusieurs captures d’écran que l’on recompose ensuite en une seule image. Pour l’instant les captures d’écran ne sont pas comparées, en cas d’écran dupliqué ils seront les deux présents.

|  |  |
| --- | --- |
| 1 | public void TakeScreenShot() |
| 2 | **{** |
| 3 | int TotalWidth **=** 0**;** |
| 4 | int MaxHeight **=** 0**;** |
| 5 | List**<**Bitmap**>** images **=** **new** List**<**Bitmap**>();** |
| 6 | //prend une capture d'écran de tout les écran |
| 7 | **foreach(**Screen screen **in** Screen**.**AllScreens**)** |
| 8 | **{** |
| 9 | Bitmap Bitmap **=** **new** Bitmap**(**screen**.**Bounds**.**Width**,** screen**.**Bounds**.**Height**,** PixelFormat**.**Format32bppArgb**);** |
| 10 | Rectangle ScreenSize **=** screen**.**Bounds**;** |
| 11 | Graphics**.**FromImage**(**Bitmap**).**CopyFromScreen**(**ScreenSize**.**Left**,** ScreenSize**.**Top**,** 0**,** 0**,** ScreenSize**.**Size**);** |
| 12 | images**.**Add**(**Bitmap**);** |
| 13 | TotalWidth **+=** ScreenSize**.**Width**;** |
| 14 | **if** **(**ScreenSize**.**Height **>** MaxHeight**){**MaxHeight **=** ScreenSize**.**Height**;}** |
| 15 | **}** |
| 16 | Bitmap FullImage **=** **new** Bitmap**(**TotalWidth**,** MaxHeight**,** PixelFormat**.**Format32bppArgb**);** |
| 17 | Graphics FullGraphics **=** Graphics**.**FromImage**(**FullImage**);** |
| 16 | int offsetLeft **=** 0**;** |
| 19 | //Crée une seul image de toutes les captures d'écran |
| 20 | **foreach** **(**Bitmap image **in** images**)** |
| 21 | **{** |
| 22 | FullGraphics**.**DrawImage**(**image**,** **new** Point**(**offsetLeft**,** 0**));** |
| 23 | offsetLeft **+=** image**.**Width**;** |
| 24 | **}** |
| 25 | //FullImage = (new Bitmap(FullImage, new Size(200,200))); |
| 26 | data**.**SetScreenShot**(**FullImage**);** |
| 27 | FullGraphics**.**Dispose**();** |
| 28 | **}** |

## Récupération des urls ouverts

Après de multiples essais la récupération de tous les onglets ouverts semble peut-être possible avec AutomationElement, mais cela serait long à apprendre et à faire pour tous les navigateurs. J’avais encore deux options : récupérer le nom de l’onglet actuel ou son url. Mais le code trouvé pour récupérer l’url était différent pour chaque navigateur et semblait ne fonctionner que sur certaines versions des navigateurs. N’importe quelle mise à jour pouvait donc demander une adaptation du code. Alors que La récupération du nom des onglets était indépendante de la version du navigateur et était identique pour tous les navigateurs. C’est pour cela que j’ai finalement choisi de ne récupérer que le nom de l’onglet ouvert actuellement.

|  |  |
| --- | --- |
| 1 | public void GetAllOpenTabName() |
| 2 | **{** |
| 3 | data**.**ClearUrls**();** |
| 4 | **foreach** **(**string singleBrowser **in** browsersList**)** |
| 5 | **{** |
| 6 | Process**[]** process **=** Process**.**GetProcessesByName**(**singleBrowser**);** |
| 7 | **if** **(**process**.**Length **>** 0**)** |
| 8 | **{** |
| 9 | **foreach** **(**Process singleProcess **in** process**)** |
| 10 | **{** |
| 11 | IntPtr hWnd **=** singleProcess**.**MainWindowHandle**;** |
| 12 |  |
| 13 | StringBuilder text **=** **new** StringBuilder**(**GetWindowTextLength**(**hWnd**)** **+** 1**);** |
| 14 | GetWindowText**(**hWnd**,** text**,** text**.**Capacity**);** |
| 15 | **if(**text**.**ToString**()** **!=** ""**){**data**.**SetUrls**(**singleBrowser **+** " : " **+** text**.**ToString**());}** |
| 16 | **}** |
| 17 | **}** |
| 18 | **}** |
| 19 | **}** |

## ClientData

Classe partagée entre les deux applications pour faciliter l’utilisation et le transfert des données. Elle contient : le nom de l’ordinateur, le nom de l’utilisateur logé, la liste des url ouverts, la liste des processus, un dictionnaire qui contient l’id et le nom des processus pour la sérialisation, la dernière capture d’écran et pour finir le socket utilisé pour communiquer avec le client. Les données avec le tag [JsonInclude] sont envoyés grâce à la sérialisation. L’image est envoyée indépendamment et les autres tagués [JsonIgnore] ne sont pas envoyés.

|  |  |
| --- | --- |
| 1 | [JsonInclude] |
| 2 | **public** string ComputerName **=** System**.**Environment**.**MachineName**;** // Le nom de l'ordinateur du client |
| 3 | **[**JsonInclude**]** |
| 4 | **public** string UserName **=** System**.**Security**.**Principal**.**WindowsIdentity**.**GetCurrent**().**Name**;** // nom d'utilisateur du pc client |
| 5 | **[**JsonInclude**]** |
| 6 | **public** List**<**string**>** Urls**;** // Liste des urls du client |
| 7 | **[**JsonInclude**]** |
| 8 | **public** Dictionary**<**int**,** string**>** DicProcesses **=** **new** Dictionary**<**int**,** string**>();** // Dictionnaire pour envoyer les id et les noms des processus |
| 9 | **[**JsonIgnore**]** |
| 10 | **private** List**<**Process**>** Processes**;** // liste des processus lancé par le client |
| 11 | **[**JsonIgnore**]** |
| 12 | **private** Bitmap ScreenShot**;** // Le dernier screenshot pris sur la machine cliente |
| 13 | **[**JsonIgnore**]** |
| 14 | **private** Socket socket**;** // Socket que l'application maître utilise pour communiquer avec ce client |

## Recevoir les demandes

Cette fonction qui tourne dans un thread parallèle au principal permet à l’application cliente de recevoir les demandes de l’application maître et de pouvoir les traiter pour savoir quelle méthode utiliser pour répondre correctement à la demande de l’application maître. Dans certains cas il y a aussi un paramètre qu’il faut analyser et passer à la méthode exécutée. Pour le moment il n’y a que 4 demandes quelle peut traiter :

Data : qui lui demande d’envoyer ces données clients.

Image : qui lui demande d’envoyer sa capture d’écran.

Kill : qui lui demande de tuer un processus reçu en paramètre.

Receive : qui lui demande de recevoir le partage d’écran de l’application maître.

|  |  |
| --- | --- |
| 1 | public void WaitForDemand() |
| 2 | **{** |
| 3 | **while(true)** |
| 4 | **{** |
| 5 | byte**[]** info **=** **new** byte**[**12**];** |
| 6 | int lenght**;** |
| 7 | **try{**lenght **=** MasterSocket**.**Receive**(**info**);}** |
| 8 | **catch(**SocketException**){return;}** |
| 9 | Array**.**Resize**(ref** info**,** lenght**);** |
| 10 | lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**Encoding**.**Default**.**GetString**(**info**));** **}));** |
| 11 | string text **=** Encoding**.**Default**.**GetString**(**info**);** |
| 12 | **switch** **(**Encoding**.**Default**.**GetString**(**info**).**Split**(**' '**)[**0**])** |
| 13 | **{** |
| 14 | **case** "data"**:**SendData**();break;** |
| 15 | **case** "image"**:**SendImage**();break;** |
| 16 | **case** "kill"**:**KillSelectedProcess**(**Convert**.**ToInt32**(**text**.**Split**(**' '**)[**1**]));break;** |
| 17 | **case** "receive"**:**Task**.**Run**(()** **=>** ScreenReceiver**());break;** |
| 18 | **case** "stop"**:**MasterSocket **=** **null;**Task**.**Run**(()** **=>** ConnectToMaster**());return;** |
| 19 | **}** |
| 20 | **}** |
| 21 | **}** |

## Arrêt

Lorsqu’elle s’arrête l’application cliente est chargée de prévenir l’application maître pour quelle ferme son socket et n’essaie plus de communiquer avec cette application cliente.

## Classe ClientForm

ClientData data : Instance de la classe ClientData pour stocker les données du client.

List<string> browserList : List des processus pour les navigateurs que l’on veut vérifier.

Socket MasterSocket : Socket pour communiquer avec l’application maître.

Socket MultiCastSocket : Socket pour la réception en multicast.

List<string> DefaultProcess : List des processus lancés lors du démarrage.

ClientForm() : Constructeur pour l’application cliente.

ScreenRecorder() : Fonction pour capturer son écran et l’envoyer au maître.

ScreeenReceiver() :Fonction pour recevoir le partage d’écran multicast.

GetDefautProcess() :Fonction pour obtenir les processus lancés au démarrage.

GetUserProcess() :Fonction pour obtenir les processus lancés par l’utilisateur.

KillSelectedProcess() :Fonction pour tuer le processus avec son id.

TakeScreenShot() :Fonction pour prendre une capture d’écran de tous les écrans.

GetAllOpenTabName() :Fonction pour obtenir tous les urls ouverts et actifs.

ConnectToMaster() :Fonction qui connecte le client à l’application maître.

WaitForDemand() :Fonction qui démarre des threads à la demande du maître.

SendData() :Fonction pour envoyer les données au maître.

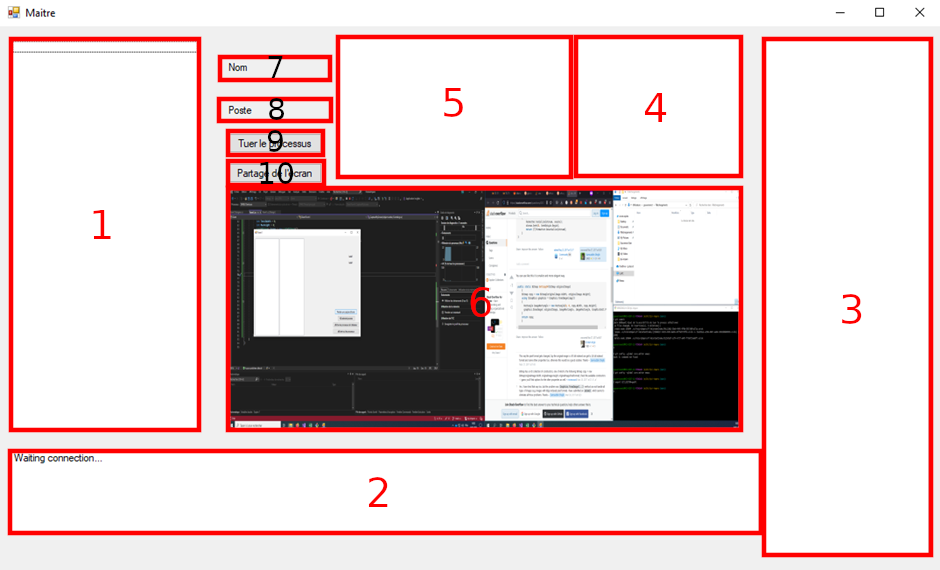
SendImage() :Fonction pour envoyer l’image au maître.

ClientClosing() :Fonction qui avertit le maître que l’application se ferme.

ClientResize() :Fonction qui affiche/cache le TrayIcon quand la fenêtre change.

IconClientClick() :Affiche le fenêtre lorsque l’on clique sur le TrayIcon.

# Application Maître



## Interface

1 : La listbox lbxClients : Affiche tous les clients connectés.

2 : La listbox lbxConnexion : Affiche les nouvelles connexions à l’application maître.

3 : La listbox lbxRequetes : Affiche les demandes de données au client ainsi que les réceptions de données.

4 : La listbox lbxProcesses : Affiche les processus du client sélectionné.

5 : La listbox lbxUrls : Affiche les urls ouverts du client sélectionné.

6 : La picturebox pbxScreenShot : Affiche la dernière capture d’écran du client sélectionné.

7 : Le label lblUserName : Affiche le nom d’utilisateur du client sélectionné.

8 : Le label lblPoste : Affiche le nom de l’ordinateur du client sélectionné.

9 : Le bouton btnKillProcess : Permet à l’utilisateur de l’application maître de tuer le processus sélectionné dans la lbxProcesses pour le client sélectionné dans lbxClients.

10 : Le bouton btnShareScreen : Permet à l’utilisateur de l’application maître de partager son écran à tous les clients connectés.

L’interface maître est complexe, elle est pour le moment encore en développement, mais elle doit permettre d’afficher toutes les informations importantes tel que les clients connectés ainsi que les détails d’un client.

## Démarrage

Au démarrage l’application maître va lancer son processus de login des clients. Elle va aussi lancer le processus de demande des données au client, mais celui-ci restera en pause tant qu’il n’y aura pas de clients.

|  |  |
| --- | --- |
| 1 | public MaitreForm() |
| 2 | **{** |
| 3 | InitializeComponent**();** |
| 4 | lbxClients**.**DataSource **=** AllClients**;** |
| 5 | //lance la boucle de connexion |
| 6 | Task**.**Run**(**LogClients**);** |
| 7 | //lance la boucle de demande des données |
| 8 | Task**.**Run**(**AskingData**);** |
| 9 | **}** |

## Résumé

L’application maître permet à un enseignant de recevoir les données des applications clientes et de les afficher. Elle permet aussi de partager l’écran de l’application maître à toutes les applications clientes. Pour finir elle peut prendre le contrôle d’une application cliente.

## Envoi de requêtes

L’application maître fait des demandes à toutes les applications clientes connectées, elle demande les données puis après les avoir reçues elle demande la capture d’écran quand elle la reçoit elle passe au prochain client. Elle boucle indéfiniment tant qu’elle a des applications clientes connectée.

|  |  |
| --- | --- |
| 1 | private void AskingData() |
| 2 | **{** |
| 3 | **while(true)** |
| 4 | **{** |
| 5 | **if(**AllClients**.**Count **!=** 0**)** |
| 6 | **{** |
| 7 | DateTime StartUpdate **=** DateTime**.**Now**;** |
| 8 | DateTime NextUpdate **=** DateTime**.**Now**.**AddSeconds**(**DurationBetweenAskingClient**);** |
| 9 | **foreach** **(**ClientData client **in** AllClients**)** |
| 10 | **{** |
| 11 | Socket socket **=** client**.**GetSocket**();** |
| 12 | socket**.**ReceiveTimeout **=** timeout**;** |
| 13 | socket**.**SendTimeout **=** timeout**;** |
| 14 | **try** |
| 15 | **{** |
| 16 | //demande les données |
| 17 | socket**.**Send**(**Encoding**.**ASCII**.**GetBytes**(**"data"**));** |
| 18 | lbxRequetes**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxRequetes**.**Items**.**Add**(**"asked data to client " **+** client**.**UserName**);** **}));** |
| 19 | Task**.**Run**(()** **=>** ReceiveData**(**client**)).**Wait**();** |
| 20 | //demande le screenshot |
| 21 | socket**.**Send**(**Encoding**.**ASCII**.**GetBytes**(**"image"**));** |
| 22 | lbxRequetes**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxRequetes**.**Items**.**Add**(**"asked image to client " **+** client**.**UserName**);** **}));** |
| 23 | Task**.**Run**(()** **=>** ReceiveImage**(**client**)).**Wait**();** |
| 24 | **}** |
| 25 | **catch** |
| 26 | **{** |
| 27 | socket**.**Shutdown**(**SocketShutdown**.**Both**);** |
| 28 | socket**.**Close**();** |
| 29 | AllClients**.**Remove**(**client**);** |
| 30 | lbxClients**.**Invoke**(new** MethodInvoker**(delegate** **{** |
| 31 | lbxClients**.**DataSource **=** **null;** |
| 32 | lbxClients**.**DataSource **=** AllClients**;** |
| 33 | **}));** |
| 34 | lbxRequetes**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxRequetes**.**Items**.**Add**(**"Le client " **+** client**.**UserName **+** "à été retiré"**);** **}));** |
| 35 | **}** |
| 36 | **}** |
| 37 | DateTime FinishedUpdate**=** DateTime**.**Now**;** |
| 38 | TimeSpan UpdateDuration **=** FinishedUpdate **-** StartUpdate**;** |
| 39 | TimeSpan CycleDuration **=** NextUpdate **-** StartUpdate**;** |
| 40 | **if** **(**CycleDuration **>** UpdateDuration**)** **{** |
| 41 | lbxRequetes**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxRequetes**.**Items**.**Add**(**"Attente du prochain cycle dans " **+** **(**CycleDuration **-** UpdateDuration**)** **+** " secondes"**);** **}));** |
| 42 | Thread**.**Sleep**(**CycleDuration**-**UpdateDuration**);** |
| 43 | **}** |
| 44 | **}else{**Thread**.**Sleep**(**100**);}** |
| 45 | **}** |
| 46 | **}** |

## Affichage des données client

L’application maître doit permettre deux vues. Une avec tous les écrans des ordinateurs clients qui permet d’avoir une vue d’ensemble sur totalité des ordinateurs clients. Et une deuxième plus précise qui affiche toutes les informations sur un seul client, et permet de d’arrêter un processus chez ce client.

## Arrêt

Lorsqu’elle s’arrête l’application maître doit prévenir toutes les applications clientes connectées pour qu’elles ferment leurs sockets et essaye de se reconnecter à nouveau comme au démarrage.

|  |  |
| --- | --- |
| 1 | private void MaitreClosing(object sender, FormClosingEventArgs e) |
| 2 | **{** |
| 3 | **foreach(**ClientData client **in** AllClients**)** |
| 4 | **{** |
| 5 | client**.**GetSocket**().**Send**(**Encoding**.**ASCII**.**GetBytes**(**"stop"**));** |
| 6 | client**.**GetSocket**().**Shutdown**(**SocketShutdown**.**Both**);** |
| 7 | client**.**GetSocket**().**Close**();** |
| 8 | **}** |
| 9 | **}** |

## Classe MaitreForm

Int DurationBetweenAskingClient : La durée entre les demandes à un client.

Int : timeout : La durée après lequel un message est perdu.

List<ClientData> : Tous les clients connectés au maître.

MaitreForm() :Constructeur pour l’application maitre.

RecordScreen() :Fonction pour capturer son écran et le partager en multicast.

LogClients() :Fonction pour enregistrer les nouveaux clients.

AskingData() :Fonction qui demande les données aux clients.

ReceiveData() :Fonction qui reçoit les données des clients.

ReceiveImage() :Fonction qui reçoit les images des clients.

AskKillProcess() :Fonction qui demande à l’application client de tuer un processus

bntShareScreen\_Click() :Fonction pour lancer ou non le partage d’écran en multicast.

DisplayClientInfo() :Fonction pour afficher les informations du client sélectionné.

MaitreClosing() :Fonction qui avertit tous les clients que l’application se ferme.

MaitreResize() :Fonction qui affiche/cache le TrayIcon quand la fenêtre change.

IconMaitreClick() :Affiche le fenêtre lorsque l’on clique sur le TrayIcon.

# Liaisons entre les deux applications

## Connexion

La connexion entre les deux applications se fait par une requête de l’application cliente à l’application maître. Pour cela l’application cliente doit connaitre l’adresse IP de l’application maître auquel elle doit se connecter. Cette connexion est utile pour créer un socket qui permet la connexion entre les deux applications à tout moment. C’est pour cela que l’application maître écoute pour des requêtes de connexion client en continu.

|  |  |
| --- | --- |
| 1 | public void LogClients() |
| 2 | **{** |
| 3 | // Establish the local endpointfor the socket. |
| 4 | // Dns.GetHostName returns the name of the host running the application. |
| 5 | IPHostEntry ipHost **=** Dns**.**GetHostEntry**(**Dns**.**GetHostName**());** |
| 6 | IPAddress ipAddr **=** ipHost**.**AddressList**[**ipHost**.**AddressList**.**Length **-**1**];** |
| 7 | IPEndPoint localEndPoint **=** **new** IPEndPoint**(**ipAddr**,** 11111**);** |
| 8 | // Creation TCP/IP Socket using Socket Class Constructor |
| 9 | Socket listener **=** **new** Socket**(**ipAddr**.**AddressFamily**,**SocketType**.**Stream**,** ProtocolType**.**Tcp**);** |
| 10 | // Using Bind() method we associate a network address to the Server Socket |
| 11 | // All client that will connect to this Server Socket must know this network Address |
| 12 | listener**.**Bind**(**localEndPoint**);** |
| 13 | // Using Listen() method we create the Client list that will want to connect to Server |
| 14 | listener**.**Listen**(-**1**);** |
| 15 | **while** **(true)** |
| 16 | **{** |
| 17 | lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**"Waiting connection..."**);** **}));** |
| 18 | **try{** |
| 19 | // Suspend while waiting for incoming connection Using Accept() method the server will accept connection of client |
| 20 | Socket clientSocket **=** listener**.**Accept**();** |
| 21 | lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**"New connexion to " **+** clientSocket**.**RemoteEndPoint**);** **}));** |
| 22 | AllClients**.**Add**(new** ClientData**(**clientSocket**));** |
| 23 | lbxClients**.**Invoke**(new** MethodInvoker**(delegate** **{** |
| 24 | lbxClients**.**DataSource **=** **null;** |
| 25 | lbxClients**.**DataSource **=** AllClients**;** |
| 26 | **}));** |
| 27 | **}** |
| 28 | **catch** **(**Exception e**){**lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**e**.**ToString**());** **}));}** |
| 29 | **}** |
| 30 | **}** |

## Socket

Un socket permet d’établir une connexion permanente entre les applications, ils nous serviront à transmettre les données sous forme de tableau de byte entre l’application cliente et l’application maître. Au moment où l’une des applications s’arrête, elle doit prévenir l’autre pour qu’elle arrête son socket.

## Données

Les données ne peuvent pas être transférées tel quel, L’application cliente doit donc sérialiser les données cliente contenue dans ClientData avant de les transformer en un tableau de byte qui est la seule forme de donné qui peut être envoyé dans le socket. Á la réception, l’application maître devra retransformer le tableau de byte reçu en une chaine de caractères qu’il pourra ensuite désérialiser pour recréer la classe ClientData.

|  |  |
| --- | --- |
| 1 | private void SendData() |
| 2 | **{** |
| 3 | GetUserProcess**();** |
| 4 | GetAllOpenTabName**();** |
| 5 | //serialization |
| 6 | string jsonString **=** JsonSerializer**.**Serialize**(**data**,** **new** JsonSerializerOptions **{** IncludeFields **=** **true,** **});** |
| 7 | //envoi |
| 8 | MasterSocket**.**Send**(**Encoding**.**ASCII**.**GetBytes**(**jsonString**),** Encoding**.**ASCII**.**GetBytes**(**jsonString**).**Length**,** SocketFlags**.**None**);** |
| 9 | **}** |

## Images

L’utilisation de SendFile() pour envoyer une image semblait être une bonne idée mais finalement il était plus simple de transformer le fichier en un tableau de byte et de l’envoyer ainsi. Il suffit simplement à l’application maître de recréer une image à partir du tableau de byte.

|  |  |
| --- | --- |
| 1 | private void SendImage() |
| 2 | **{** |
| 3 | TakeScreenShot**();** |
| 4 | byte**[]** image**;** |
| 5 | ImageConverter converter **=** **new** ImageConverter**();** |
| 6 | image **=** **(**byte**[])**converter**.**ConvertTo**(**data**.**GetScreenShot**(),** **typeof(**byte**[]));** |
| 7 | MasterSocket**.**Send**(**image**,**0**,**image**.**Length**,**SocketFlags**.**None**);** |
| 8 | **}** |

## Flux d’image

Le flux d’image permet un partage vidéo de l’écran de l’application maître à toutes les applications clientes. Pour cela elle doit prendre une capture d’écran puis la séparer un plusieurs paquets pour pouvoir l’envoyer.

|  |  |
| --- | --- |
| 1 | public void RecordScreen() |
| 2 | **{** |
| 3 | **foreach** **(**ClientData client **in** AllClients**)** |
| 4 | **{**client**.**GetSocket**().**Send**(**Encoding**.**ASCII**.**GetBytes**(**"receive"**));}** |
| 5 | **using** **(**var udpClient **=** **new** UdpClient**(**AddressFamily**.**InterNetwork**))** |
| 6 | **{** |
| 7 | var address **=** IPAddress**.**Parse**(**"224.168.55.25"**);** |
| 8 | var ipEndPoint **=** **new** IPEndPoint**(**address**,** 11112**);** |
| 9 | udpClient**.**JoinMulticastGroup**(**address**);** |
| 10 | int headerSize **=** 5**;** |
| 11 | **while** **(true)** |
| 12 | **{** |
| 13 | Screen screen **=** Screen**.**AllScreens**[**1**];** |
| 14 | Bitmap bitmap **=** **new** Bitmap**(**screen**.**Bounds**.**Width**,** screen**.**Bounds**.**Height**,** PixelFormat**.**Format32bppArgb**);** |
| 15 | Rectangle ScreenSize **=** screen**.**Bounds**;** |
| 16 | Graphics**.**FromImage**(**bitmap**).**CopyFromScreen**(**ScreenSize**.**Left**,** ScreenSize**.**Top**,** 0**,** 0**,** ScreenSize**.**Size**);** |
| 17 | ImageConverter converter **=** **new** ImageConverter**();** |
| 18 | byte**[]** imageArray **=** **(**byte**[])**converter**.**ConvertTo**(**bitmap**,** **typeof(**byte**[]));** |
| 19 | pbxScreenShot**.**Invoke**(new** MethodInvoker**(delegate** **{** pbxScreenShot**.**Image **=** bitmap**;** **}));** |
| 20 | int messageLength **=** 65000**;** |
| 21 | lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**imageArray**.**Length**);** **}));** |
| 22 | **for** **(**int i **=** 0**;** i **<** imageArray**.**Length **/** **(**messageLength **-** headerSize**)** **+** 1**;** i**++)** |
| 23 | **{** |
| 24 | byte**[]** message **=** **new** byte**[**messageLength**];** |
| 25 | string strHeader **=** "datas"**;** |
| 26 | int size **=** messageLength**;** |
| 27 | **if** **(**i **==** 0**)** **{** strHeader **=** "start"**;** **}** |
| 28 | **if** **(**i **>=** imageArray**.**Length **/** messageLength**)** **{** strHeader **=** "ended"**;** **}** |
| 29 | **for** **(**int j **=** 0**;** j **<** headerSize**;** j**++)** **{** message**[**j**]** **=** Encoding**.**ASCII**.**GetBytes**(**strHeader**)[**j**];** **}** |
| 30 | **if** **(**strHeader **==** "ended"**){** |
| 31 | **for** **(**int j **=** 0**;** j **<** imageArray**.**Length **%** **(**messageLength **-** headerSize**);** j**++){** |
| 32 | message**[**j**+** headerSize**]** **=** imageArray**[**i **\*** **(**messageLength **-** headerSize**)** **+** j**];** |
| 33 | **}** |
| 34 | Array**.**Resize**(ref** message**,** imageArray**.**Length **%** messageLength **-** headerSize**);** |
| 35 | size **=** message**.**Length**;** |
| 36 | **}** |
| 37 | **else{** |
| 38 | **for** **(**int j **=** 0**;** j **<** messageLength**-** headerSize**;** j**++)** |
| 39 | **{**message**[**j**+** headerSize**]** **=** imageArray**[**i **\*** **(**messageLength**-** headerSize**)** **+** j**];}** |
| 40 | **}** |
| 41 | udpClient**.**Send**(**message**,**size**,**ipEndPoint**);** |
| 42 | **}** |
| 43 | **}** |
| 44 | **}** |
| 45 | **}** |

L’application cliente doit recevoir tous ces paquets puis recomposer une image avec puis l’afficher à l’utilisateur jusqu’à ce que la prochaine arrive.

|  |  |
| --- | --- |
| 1 | public void ScreenReceiver() |
| 2 | **{** |
| 3 | UdpClient udpClient **=** **new** UdpClient**(**MulticastPort**);** |
| 4 | udpClient**.**JoinMulticastGroup**(**IPAddress**.**Parse**(**"224.168.55.25"**));** |
| 5 | var ipEndPoint **=** **new** IPEndPoint**(**IPAddress**.**Any**,**0**);** |
| 6 | int headerSize **=** 5**;** |
| 7 | // Receive messages |
| 8 | **while** **(true)** |
| 9 | **{** |
| 10 | **try** |
| 11 | **{** |
| 12 | byte**[]** imageBuffer **=** **new** byte**[**1048576**];** |
| 13 | int lastId **=** 0**;** |
| 14 | string header**;** |
| 15 | **do** |
| 16 | **{** |
| 17 | byte**[]** message **=** udpClient**.**Receive**(ref** ipEndPoint**);** |
| 18 | header **=** Encoding**.**Default**.**GetString**(**message**).**Substring**(**0**,** headerSize**);** |
| 19 | **for** **(**int i **=** 0**;** i **<** message**.**Length**-** headerSize**;** i**++)** |
| 20 | **{**imageBuffer**[**i**+**lastId**]** **=** message**[**i**+** headerSize**];}** |
| 21 | lastId **+=** message**.**Length **-** headerSize**;** |
| 22 | lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**header **+**": "**+** message**.**Length**);** **}));** |
| 23 | **}** **while** **(**header **!=** "ended"**);** |
| 24 | lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**lastId**);** **}));** |
| 25 | Bitmap bitmap **=** **new** Bitmap**(new** MemoryStream**(**imageBuffer**));** |
| 26 | pbxMasterScreen**.**Invoke**(new** MethodInvoker**(delegate** **{** pbxMasterScreen**.**Image **=** bitmap**;** **}));** |
| 27 | **}** |
| 28 | **catch** |
| 29 | **{** |
| 30 | lbxConnexion**.**Invoke**(new** MethodInvoker**(delegate** **{** lbxConnexion**.**Items**.**Add**(**"l'image n'a pas étée recue"**);** **}));** |
| 31 | **}** |
| 32 | **}** |
| 33 | **}** |

## Demande d’arrêt d’un processus

L’application maître peut demander à une application cliente sélectionnée d’arrêter un processus sélectionné que l’utilisateur de l’application maître aurait jugé indésirable.

|  |  |
| --- | --- |
| 1 | private void AskKillProcess(object sender, EventArgs e) |
| 2 | **{** |
| 3 | ClientData client **=** lbxClients**.**SelectedItem **as** ClientData**;** |
| 4 | **if** **(**lbxProcesses**.**SelectedItem **==** **null)** **{** **return;** **}** |
| 5 | int id **=** client**.**DicProcesses**.**FirstOrDefault**(**x **=>** x**.**Value **==** lbxProcesses**.**SelectedItem**.**ToString**()).**Key**;** |
| 6 | client**.**GetSocket**().**Send**(**Encoding**.**ASCII**.**GetBytes**(**"kill "**+**id**));** |
| 7 | lbxRequetes**.**Items**.**Add**(**"killed processus " **+** lbxProcesses**.**SelectedItem**.**ToString**());** |
| 8 | lbxRequetes**.**Items**.**Remove**(**lbxRequetes**.**SelectedItem**);** |
| 9 | **}** |

# Tests

## Méthode de test

Les tests ont été créés et effectués avec des tests unitaires en C#. Cela me permet d’automatiser tous les tests et de vérifier si mes changements ont affecté les résultats en temps réel. J’ai donc créé 3 projets de test, un pour l’application maître, un pour l’application cliente qui me permet de tester toutes les récupérations de donnée et un dernier pour tester les échanges entre les deux applications.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Catégorie, critères de test | Résultats attendus | Succès Ok/Ko |
| **Application Cliente** | | | |
| 1.1 | Nouveau processus | Test Unitaires | Ok |
| 1.2 | Tuer un processus par id | Test Unitaires | Ok |
| 1.3 | Prendre une capture d'écran | Test Unitaires | Ok |
| 1.4 | Obtenir tous les url | Test Unitaires | Ok |
| 1.5 | Minimisation en TrayIcon | Test Unitaires | Ok |
| **Application maître** | | | |
| 2.1 | Minimisation en TrayIcon | Test Unitaires | Ok |
| 2.2 | Affichage du client sélectionné | Visuel | Ok |
| 2.3 | Arrêt du processus demandé | Visuel | Ok |
| 2.4 | Partage de l'écran | Visuel | Ko |
| **Communication / Transferts de données** | | | |
| 3.1 | Connexion au client | Test Unitaires | Ok |
| 3.2 | Connexion au maître | Test Unitaires | Ok |
| 3.3 | Connexion multi-client | Test Unitaires | Ok |
| 3.4 | Transfert des données | Visuel | Ok |
| 3.5 | Transfert des images | Visuel | Ok |

# Problèmes et solutions

**Connexion à l’application maître :** L’application cliente a besoin de l’adresse IP de l’application Maître pour se connecter ce qui semble impossible car il y a plusieurs applications maître dans le même sous réseau. Il y a aussi le problème que les ordinateurs maîtres n’ont pas de nom ou d’IP fixe

**Envoi des captures d’écran :** Les images ne peuvent pas simplement être envoyé sur le Thread principale car cela gèlerait l’écran jusqu’à ce que l’image soit envoyé, j’utilise donc un deuxième thread pour envoyer l’image de manière indépendante. J’aurais aussi pu envoyer l’image de manière asynchrone mais cela semblait plus difficile à faire.

**SendFile() a besoin d’un fichier sauvegardé :** La méthode SendFile() que je voulais utiliser pour envoyer les images ne prenait que le chemin à un fichier sauvegardé sur le disque en argument. En plus que de ralentir le processus cela permettait à l’utilisateur de modifier ou de supprimer la capture d’écran avant l’envoi. J’ai donc décidé d’utiliser la méthode Send() qui permet d’envoyer un tableau de byte. Il me suffisait au préalable de convertir l’image en tableau de byte. Puis de la recréer du coté maître.

**Envoi des processus lancée par l’utilisateur :** Les processus récupérer sont des Threads qu’il faut envoyer à l’application maître mais les Threads ne peuvent pas être envoyé. J’envoie donc à la place le nom du processus pour que l’utilisateur de l’application maître puisse l’identifier ainsi que son id qui sera renvoyé par le l’application maître s’il veut arrêter ce processus.

**Seul le thread principal peut modifier le formulaire :** Lorsque l’on veut modifier le formulaire depuis un autre thread on crash avec l’erreur « Opération inter-threads non valide : le contrôle XXX a fait l'objet d'un accès à partir d'un thread autre que celui sur lequel il a été créé. ». Pour éviter ce problème il faut donc remplacer la première ligne par la deuxième pour résoudre le problème.

|  |  |
| --- | --- |
| 1 | XXX**.**Name **=**"newname"**;** |
| 2 | XXX**.**Invoke**(new** MethodInvoker**(delegate{**XXX**.**name**=**"newname"**;}));** |

**Envoi en multicast du partage d’écran maître :** Le multi casting n’est pas facile à réaliser car en plus de devoir envoyer les données à plusieurs clients cette procédure se complique car l’utilisation d’un socket de type Dgram est obligatoire, mais cela limite grandement la taille maximale des paquets que l’on peut envoyer. Je dois donc Séparer les images en plusieurs paquets, les envoyer à tous les clients qui doivent ensuite recomposer une image.

# Références

## Webographie

Sérialisation et désérialisation des données :<https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to?pivots=dotnet-7-0>

Envoi de données avec des socket :<https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/sockets/socket-services>

Résolution des problèmes

Code pour récupérer le nom des onglets actuellement sélectionnés dans tous les navigateurs ouverts :<https://stackoverflow.com/questions/40434679/c-sharp-get-all-open-browsing-tabs-in-all-instances-of-firefox>

Ce tutoriel m’a permis d’avoir un code fonctionnel pour les sockets que j’ai pu modifier à mes besoins :<https://www.geeksforgeeks.org/socket-programming-in-c-sharp/>

Cette discussion m’a permis de résoudre le problème que seul le thread principale avait accès au formulaire :<https://www.developpez.net/forums/d750162/dotnet/developpement-windows/windows-forms/operation-inter-threads-non-valide/>

Cette page m’a permis d’avoir un multicast Udp fonctionnel pour du texte :  <https://stackoverflow.com/questions/23846116/udp-multicasting-c-sharp>.

## Glossaire

**Byte :** Chiffre entre 0 et 255

**Tableau :** Ensemble de taille choisie de données de même type.

**Convertir :** Changer le format des données

**Sérialiser :** Convertir les données dans un format texte json ou xml.

**Socket :** Canal de discussion entre deux applications

**Thread :** Processus

**Désérialiser :** Action inverse à la sérialisation.

# Bilan

## Améliorations possibles

Le partage d’écran de l’application maître ne fonctionne pas sur tous les ordinateurs clients.

Les images reçues du partage de l’application maître sont parfois blanche en bas.

Permettre d’ouvrir une autre fenêtre pour un client tout en gardant la vue d’ensemble.

Améliorer les vérifications de reçu des messages.

Processus de reconnexion à la même IP la semaine d’après à la même heure.

## Objectifs atteints

### Client

Prendre une capture d’écran de tous les écrans

Récupération des processus lancé par l’utilisateur.

Récupération de tous les urls ouvert dans tous les navigateurs

Minimiser en TrayIcon pour cacher l’application.

### Maître

Arrête des processus indésirables des clients

Minimiser en TrayIcon pour cacher l’application.

Afficher les données reçues.

Afficher les captures d’écrans reçues.

### Communication

Envoi des données dans des threads séparé.

Transférer les données à haut débit.

Partage d’écran de l’application maître semi terminé.

## Conclusion

Bien que tout ne soit pas fonctionnel les fonctionnalité présentes sont bien implémentées et stables. Malgré des difficultés tel que l’apprentissage pour transférer des données sur le réseau ou pour envoyer des images en multicast à tous les clients. Elles ne m’ont pas arrêté et j’ai réussi à rendre un produit dont je suis fière. Malgré que le code ne soit pas parfait, je pense qu’il représente bien le meilleur de ce que je puisse faire dans la limite de temps qu’il m’était accordé.

Les difficultés rencontrées durant ce projet m’ont appris beaucoup de chose sur des sujets très variés et intéressants tel que l’envoi et la réception de données sur le réseau, la sérialisation des données ainsi que leur désérialisation et la création d’une interface qui soit ergonomique et adaptée aux données à présenter. Toutes ces brides de savoir m’aiderons très certainement dans la réalisation de mon TPI.

# Annexe Websummary

**Remplacement d’Impero**

**Gouvernon Stan**, canton Neuchâtel, CPNE-TI

**Environnement**

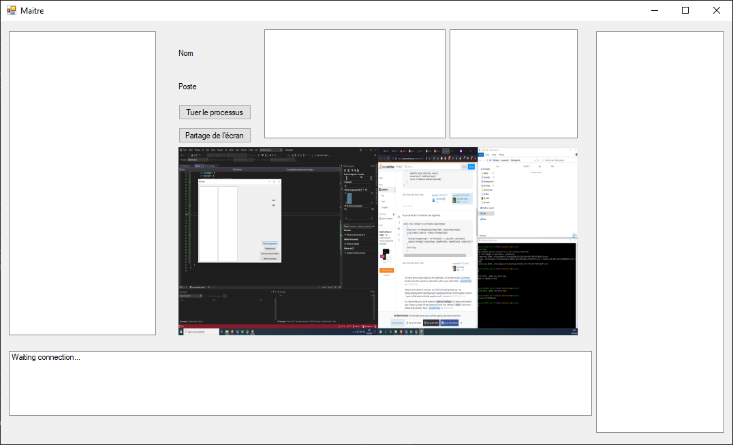
Étudiant au CPNE-TI depuis maintenant 3 ans, je me suis habitué à l’utilisation du logiciel Impero. Il permet aux enseignant de monitorer leur classe, de faire des présentations à toute la classe et d’aider un élève.

**Situation de départ**

Mais Impero présente maintenant un défaut majeur : il doit être installé sur les machines. Ce qui ne posait pas de problème avant va désormais en poser un car l’école va dans quelques années demander aux élèves de prendre leurs propres ordinateurs portables. L’école ne pourra donc pas obliger un élève à installer Impero. Il faut donc que je crée une alternative portable.

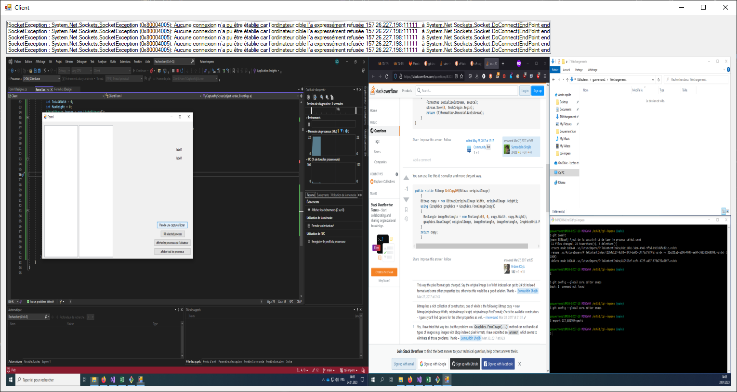
**Objectifs**

Mon objectif durant ce TPI est de créer deux application (professeur/élève) qui permettent de reproduire les fonctionnalités d’Impero. Tel que l’envoi de donnée des élèves au professeur ou le partage d’écran à tous les élèves. Il me faudra aussi planifier, tenir un journal de travail et rendre un rapport complet sur le sujet.

**Représentation graphique**

Vous avez à droite les deux applications en haut l’application du professeur qui permet l’affichage de toutes les informations sur les élèves. Et en bas l’application élève qui permet de recevoir le partage d’écran.

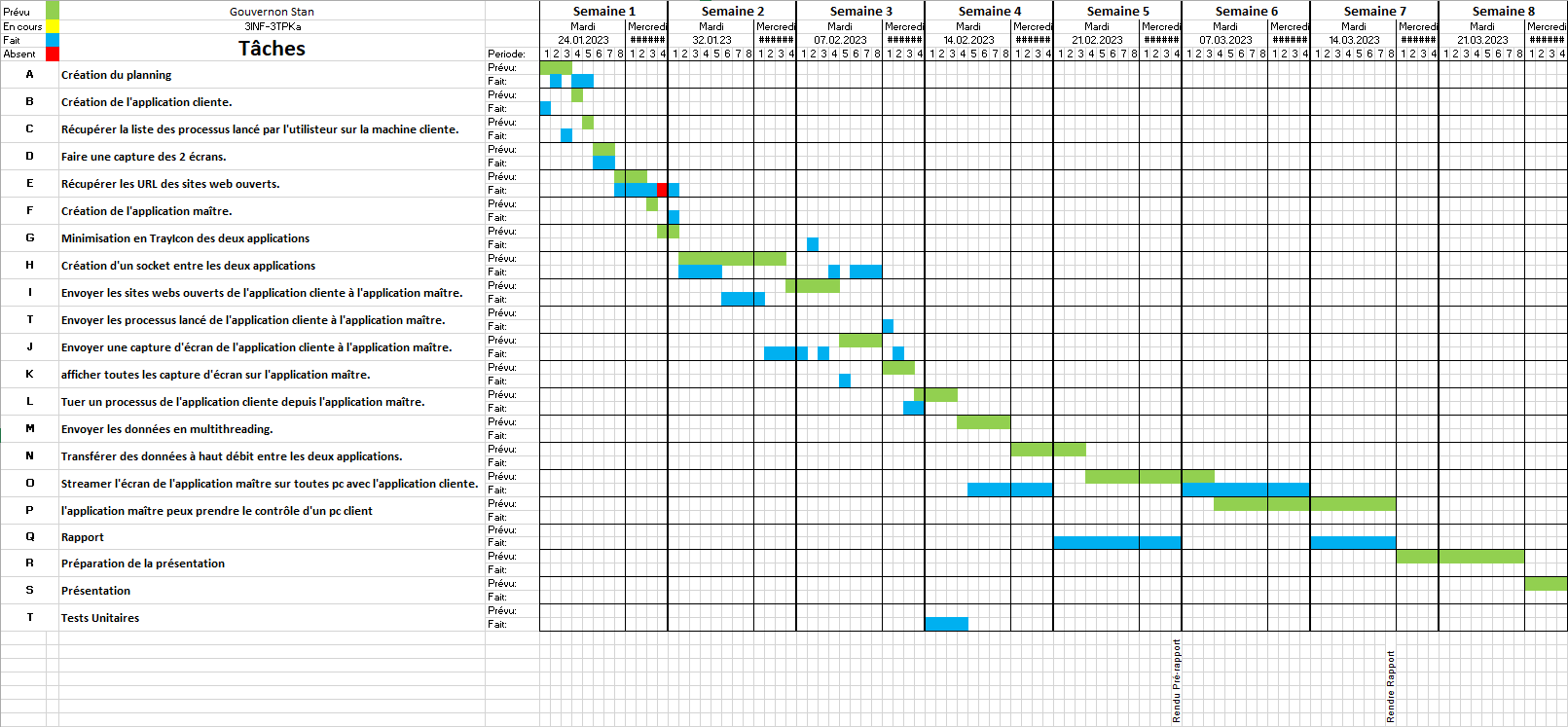
**Voie Suivie**

L’application devant servir aux élèves et aux professeurs il à fallut durant tout le développement que l’application soit utile pour les deux partis. Le développement c’est fait en C# car c’est le langage qui est enseigné dans l’école et il permet le développement d’applications graphiques.

**Résultat**

Malgré un résultat qui ne peux pas être considéré comme totalement terminé, ce qui est présent fonctionne et permet déjà son utilisation. Malgré que des améliorations de l’affichage pour le rendre plus joli semble une priorité.

# Annexe Planification



# Annexe Manuel d’utilisateur

## Application Cliente

L’application cliente n’est pas utile à l’élève tant qu’il n’a pas de partage d’écran. Il peut donc la minimiser en TrayIcon en attendant le partage.

## Application Maître

L’application maître est plus complexe mais les actions utilisateur sont limitées, il n’y a donc que 3 actions possibles

1. Afficher les données de l’élève sélectionné dans la liste 1
2. Arrêter un processus sélectionné dans la liste 4 pour l’élève sélectionné dans la liste 1
3. Partager son écran à tous ces élèves



# Annexe Journal de travail

Mardi 24.01.2023

Objectifs A : Planification B : Création de l’application cliente

C : Lister les processus C : Tuer les processus indésirables

D : Screenshot des deux écrans E : Récupérer les urls ouverts

Durée 8 périodes

Déroulement

8h35-9h20 : Mise en place de git et création du projet.

9h20-9h50 : Début du planning.

9h50-10h30 : Récupération de la liste des processus actuels.

10h30-11h20 : Comparaisons avec la liste des processus déjà lancés.

11h20-11h50 : Suite du planning.

12h40-13h10 : Fin du planning.

13h10-14h40 : Screenshot des deux écrans.

14h40-15h55 : Obtenir les urls ouvert

Problèmes / solutions

* Obtenir les urls ouverts.
* Pas encore de solutions.

Synthèse

J’ai débuté par créer la solution et le projet pour l’application cliente, j’ai commencé la planification qui fut interrompue car je n’avais pas les dates des livrables, j’ai donc poursuivi par la récupération des processus lancés par l’utilisateur. Après cela je suis retourné à ma planification pour la terminer. Ensuite j’ai poursuivi le développement en prennent une capture d’écran. Je suis ensuite arrivé à mon premier problème obtenir les urls ouverts par l’utilisateur.

Tâches

A-D 100%

E 25%

Mercredi 25.01.2023

Objectifs E : Récupérer les urls ouverts

Durée 3 périodes

Déroulement

12h40-13h00 : Rapport pour les étapes faites le jour d’avant.

13h00-13h15 : Ajout de commentaire dans le code.

13h15-15h00 : Récupération des urls ouverts

Problèmes / solutions

* Récupérer les urls ouverts dans tous les navigateurs.
* <https://stackoverflow.com/questions/40434679/c-sharp-get-all-open-browsing-tabs-in-all-instances-of-firefox> Permet de récupérer le nom de l’onglet actif de tous les browsers. Mais pas l’url ni les autres onglets
* D’autre code permette de récupérer l’url de l’onglet actif mais pas les autres. Et ils sont différentes pour chaque browser. Exemple : <https://stackoverflow.com/questions/7814027/how-can-i-get-urls-of-open-pages-from-chrome-and-firefox?noredirect=1&lq=1>

Synthèse

J’ai commencé par faire la partie du rapport que je n’avais pas eu le temps de faire le mardi 24.01. J’ai ensuite commencé les commentaires pour la documentation. Puis j’ai enfin pus continuer l

Tâches

E 50 %

Mardi 31.01.2023

Objectifs H : Création d’un socket entre les applications

I : Envoi des données. E : Obtenir le nom des onglet actifs

Durée 8 périodes

Déroulement

8h35-9h00 : Écriture du rapport sur les problèmes rencontrés sur la tâche E.

9h00-9h30 : Obtention du nom des onglets actifs.

9h30-10h00 : Lecture de la doc et apprentissage sur les sockets.

10h00-11h50 : Début du développement du socket.

12h40-13h30 : Suite du socket.

13h30-14h20 : Création de la classe ClientData.

14h20-15h50 : Envoi des données réussit.

Problèmes / solutions

* Aucune connaissance sur les sockets.
* <https://www.geeksforgeeks.org/socket-programming-in-c-sharp/> M’a permis d’avoir un code fonctionnel à modifier.
* <https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/sockets/socket-services> M’a permis d’apprendre la théorie.
* Aucune connaissance sur la sérialisation.
* <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to?pivots=dotnet-7-0>
* L’application Maître ne peux pas faire de demande explicite.
* Possible : Envoi d’un message inutile par l’application maitre que servirait de signal à l’application cliente.

Synthèse

J’ai commencé par écrire les raisons et la solution à la tâche E dans mon rapport. Puis j’ai terminé d’implémenter la solution trouvée. La tache H qui est l’implémentation du socket entre l’application maître et l’application cliente a été avancée car elle était trop cruciale pour être laissé pour plus tard. Je me suis donc documenté à ce sujet puis j’ai commencé à l’implémenter. J’ai Crée la classe ClientData qui va me permettre d’envoyer les données plus efficacement sur le réseau. Grâce à cela j’ai réussi à envoyer les données correctement mais pas encore à les désérialiser.

Tâches

E 100%

I 85%

H : 100%

Mercredi 01.02.2023

Objectifs I : Envoi des donnée J : Envoi du Screenshot

Durée 4 périodes

Déroulement

12h40-13h10 : Résolution du problème à la désérialisation des données.

13h10-14h00 : Début de l’envoi de l’image à l’application maitre.

14h00-15h15 : Suite du transfert de l’image

15h15-15h50 : Suite de la documentation.

Problèmes / solutions

* Désérialiser les données car on reçoit un tableau de byte avec des valeurs inutiles à la fin.
* La fonction revoie le nombre de byte reçu.
* Les deux flux de données étaient reçus en même temps ce qui créait des interférences.
* Envoi de l’image qu’après confirmation de la réception des données par le maitre.
* Seul le début de l’image est reçu.
* Envoi asynchrone pour permettre à toute l’image d’arriver avant de l’utiliser.

Synthèse

J’ai commencé par résoudre le problème de la taille de du tableau du jour d’avant. Puis j’ai commencé à envoyer les Screenshot à l’application maitre, après avoir résolu les interférences entre les données et l’image un nouveau problème est apparu, l’image n’arrivait pas en entier. Je vais donc devoir implémenter des transferts asynchrones la semaine prochaine. J’ai terminé par faire le rapport qui a du retard.

Tâches

I :100%

J :60%

Mardi 07.02.2023

Objectifs J : Envoi des données en aync G : Minimisation en TrayIcon

Durée 8 périodes

Déroulement

8h35-9h20 : Recherches et documentation sur l’envoi de fichier asynchrone.

9h20-10h00 : Minimisation des deux applications en TrayIcon.

10h00-11h20 : Restructuration des communications réseau.

11h20-11h50 : Envoi de l’image.

12h40-13h00 : Affichage des données.

13g00-14h50 : Corrections de multiples bugs.

14h50-15h55 : Rendu et présentation du projet.

Problèmes / solutions

* Modifier les contrôles depuis les threads.
* <https://www.developpez.net/forums/d750162/dotnet/developpement-windows/windows-forms/operation-inter-threads-non-valide/>
* lbxConnexion.Invoke(new MethodInvoker(delegate{lbxConnexion.Items.Add()}));
* Envoyer l’image à l’application maitre.
* Ne pas utiliser SendFile() mais convertir l’image en byte[] puis l’envoyer avec Send().
* Envoyer des threads avec Serialize().
* Les threads sont bloqués, ils ne peuvent pas être sérialisé j’envoie donc leur nom.

Synthèse

J’ai commencé à me documenter sur les envois asynchrones malgré que cela se révélera inutile car j’ai résolu le problème avec du multithreading. Puis j’ai commencé à développer avec facilité la minimisation en TrayIcon des deux applications. Mon code d’envoi et de réception des données n’était pas claire et pas structuré. Je l’ai donc refait de zéro pour mieux comprendre ce que je faisais, cela m’a permis de résoudre l’envoi des images. Puis j’ai affiché les données sur l’application maitre. Pour finir j’ai résolu de multiples bugs pour le rendu. J’ai ensuite fait une petite démo.

Tâches

G : 100%

J : 100%

Mercredi 08.02.2023

Objectifs Envoi des processus Amélioration et commentaires

L : Arrêter les processus à distance

Durée 4 périodes

Déroulement

12h40-13h00 : Envoi des processus dans un dictionnaire.

13h00-13h30 : Ajout de commentaires.

13h30-14h00 : Écriture du rapport.

14h00-14h20 : Envoi des images sans les sauver sur le disque.

14h20-15h30 : Terminer les processus à distance

Problèmes / solutions

* Les images devaient être sauvée sur le disque car SendFile() n’utilisait que des fichiers sauvegardés.
* Je n’utilise plus SendFile() Je peux donc convertir l’image en tableau de byte et l’envoyer.

Synthèse

J’ai commencé par résoudre le problème de l’envoi des processus qui n’est pas possible mais il est possible d’envoyer le nom et l’id des processus dans un dictionnaire. J’ai ensuite écrit tous les commentaires pour le Mardi 07.02.23. J’ai ensuite pu améliorer l’envoi des images en évitant de les sauver sur le disque. J’ai terminé la journée en arrêtant à distance les processus indésirables.

Tâches

L :80%

Mardi 14.02.2023

Objectifs Vérifications Tests unitaires

O : Stream de l’écran Maître

Durée 8 périodes

Déroulement

8h35-8h50 : Créations des deux projets de tests unitaires.

8h50-10h10 : Ajout de Try Catch pour empêcher les crashs.

10h10-10h20 : Test de connexion de 8 clients en même temps.

10h20-11h30 : Création de tests pour l’application cliente.

11h30-11h50 : Création de tests pour la connexion entre applications.

12h40-13h20 : Recherches pour le partage d’écran.

13h20-14h50 : Début de la capture vidéo avec des captures d’écrans.

Problèmes / solutions

* Les clients se connectent sur le même port que l’envoi des données.
* Ce n’est pas un problème car il n’y a pas de collisions.
* Prendre beaucoup de screenshot et les afficher prend toute la mémoire.
* Il ne suffit pas de disposer de l’image il faut aussi disposer de Graphics.

Synthèse

Maintenant que le projet à des bases fonctionnelles j’ai décidé de commencer à implémenter des tests unitaires pour valider ce que j’avais déjà fait. J’ai aussi ajouté des Try Catch dans le code pour éviter que les applications crash par exemple lorsque l’autre application s’arrête ou que le socket se ferme. J’ai ensuite créé les tests de l’application cliente il s’agit principalement de test sur la récupération des données du client. J’ai ensuite fait des tests pour la connexion des clients à l’application maître. J’ai poursuivi par des recherches sur la manière de capturer un écran et de l’envoyer sur le réseau puis d’en refaire une vidéo.

Tâches

S :40%

O :25%

Mercredi 15.02.2023

Objectifs O : Stream de l’écran Maître

Durée 4 périodes

Déroulement

12h40-13h40 : Envoi de texte en multi casting.

13h40-15h55 : Tentatives d’envoi d’images en multi casting

Problèmes / solutions

* Faire du multi casting.
* <https://stackoverflow.com/questions/23846116/udp-multicasting-c-sharp>
* Envoyer des images en multi casting.
* Pas encore de solution, les images sont trop grosses.
* <https://social.msdn.microsoft.com/Forums/en-US/fb082e81-06a0-4e3b-a54a-fa9553022b4f/does-any-way-to-increase-the-buffer-of-datagram-messages?forum=netfxnetcom>

Synthèse

Après des recherches pour Faire du multicast et avoir commencé à l’implémenter je vais utiliser le protocole Udp qui semble être le seul qui puisse faire du multi casting. Pour le type de transfert j’ai le choix entre : Dgram mais les messages ne sont pas certains d’arriver ni dans l’ordre que j’ai réussi à faire fonctionner ou Rdm qui assure que les messages arrivent et dans l’ordre mais je n’ai pas réussi à le faire fonctionner. Mais Udp ne permettent pas d’envoyer de gros messages il va donc surement falloir envoyer les messages de manières asynchrone ou les séparer en plus petits messages.

Tâches

O :60%

Mardi 21.02.2023

Objectifs Q : Amélioration du pré-rapport Commentaires et améliorations

Durée 8 périodes

Déroulement

7h50-9h50 : Écriture et ajout de code dans le pré-rapport.

9h50-10h40 : Ajout et mise au jour de commentaires dans le code.

10h40-11h50 : Suite de l’écriture du rapport.

12h40-14h10 : Code pour permettre de voir les données du client sélectionné.

14h10-15h55 : Fin de l’écriture du rapport pour aujourd’hui.

Problèmes / solutions

* Afficher le code dans le rapport.
* M.Hêche nous a donné une technique.

Synthèse

Étant donné que le pré-rapport doit être rendu demain j’ai décidé de consacrer cette journée à la documentation. Car jusqu’à maintenant je n’avais fait que noter ce qui était important et ce que je devais me rappeler. Il fallait donc développer ainsi que tenter de vulgariser un minimum mon travail. En plus de cela j’ai ajouté des images dans le rapport ainsi que des bouts de code. Je n’ai tout de même pas résisté à faire un peu de développement c’est pourquoi j’ai ajouté la possibilité de sélectionner un utilisateur pour voir ses informations. Puis j’ai continué à avancer mon rapport.

Mercredi 22.02.2023

Objectifs Q : Finir le pré-rapport

Durée 4 périodes

Déroulement

12h40-14h30 : Suite et fin du pré-rapport.

14h30-15h20 : Séparation de la logique du code de l’interface.

15h20-15h55 : Finalisation et rendu du pré-rapport.

Problèmes / solutions

* Faire de beaux diagrammes pour le pré-rapport.
* J’ai utilisé Mermaid.

Synthèse

Malgré que la journée fût majoritairement dévouée pour le pré-rapport, j’ai tout de même remarqué un gros problème dans mon code la logique n’était pas séparée de l’interface ce qui posait de gros problèmes pour les tests unitaires. J’ai donc commencé à résoudre ce problème avant de finir et de rendre le pré-rapport.

Mardi 07.03.2023

Objectifs O : Multicast Correction pré-rapport

Durée 8 périodes

Déroulement

8h35-9h00 : Reçu du pré-rapport.

9h00-9h30 : Corrections minimes dans le rapport.

9h30-11h50 : Suite du streaming en multicast.

12h40-14h20 : Tentative de résolutions des problèmes de streaming.

14h20-15h50 : Rapport et journal de travail.

Problèmes / solutions

* Les images sont trop grosses pour être envoyées avec du multicast.
* Je les divise pour les envoyer et les recompose à l’arrivé.
* Seul le haut de l’écran est affiché sur les applications clientes
* Non : L’affichage est ralenti par l’appel avec Invoke(new MethodeInvoker(delegate{})) ;
* Non : Les images sont corrompues.
* Non : Les images n’arrivent pas complètes.
* Le multi casting ne fonctionne pas sur les autres ordinateurs mais fonctionne en local.

Synthèse

Nous avons commencé par recevoir et discuter de la note ainsi que des commentaires sur le pré-rapport. J’ai ensuite fait des petites modifications dans le rapport pour le rendre meilleur d’après les remarques reçues. J’ai ensuite repris le streaming de l’écran de l’application maître en multicast à toutes les application clientes. J’ai réussi à le faire fonctionner sur de petites images et seulement en local durant la matinée. Le début d’après-midi fut dédié à résoudre ces bugs malgré que cela ne fut pas très fructueux. Pour finir la journée je suis retourné sur mon rapport.

Tâches

O :80%

Mercredi 08.03.2023

Objectifs Nettoyage du code O : Finir le streaming en multicast

Durée 4 périodes

Déroulement

12h40-14h00 : Envoi correct des images décomposées.

14h00-14h40 : Tests : le multicast ne fonctionne pas sur tous les PC.

14h40-15h50 : Nettoyage et améliorations.

Problèmes / solutions

* Seul le haut de l’écran est affiché sur les applications clientes
* Problème dans mon code d’envoi (décalage des données à cause de mes header).
* Seul certains PC reçoivent la diffusion en multicast.
* Non : Le multicast ne fonctionne pas.
* Non : Le port utilisé est fermé.
* Non : L’application à besoin de droits administrateurs.
* Les données ne sont pas reçues pour une raison encore inconnue.

Synthèse

Après avoir appris de mes essais de hier, j’ai finalement réussi à faire fonctionner le streaming en multicast. Mais malgré cela la réception des paquets ne fonctionne pas sur tous les PC. J’ai donc fait différents tests pour identifier la différence entre les PC qui fonctionne et ceux qui ne fonctionnent pas mais il n’y a pour l’instant aucun résultats concluant. J’ai donc terminé ma journée en nettoyant mon code qui en avait grandement besoin, ainsi que réécrire les bouts de code qui le devaient.

Tâches

O :95%

Mardi 14.03.2023

Objectifs Q : Corriger le rapport final Améliorer le code

Durée 8 périodes

Déroulement

8h35-11h00 : Améliorations du rapport d’après les corrections.

11h00-11h50 : Améliorations du code d’après les corrections.

12h40-14h00 : Ajouts des annexes.

14h00-15h15 : Essais pour l’affichage individuel des élèves.

Problèmes / solutions

* Pas de vrai problème rencontré.

Synthèse

J’ai décidé de commencer les corrections par le rapport puis de faire des corrections dans le code. L’après-midi j’ai commencé par rajouter les annexes manquantes tel que le Websummary, la planification et le manuel utilisateur. J’ai fini l’après-midi en peaufinant le code. Et en essayant d’ajouter un affichage individuel dans l’application maître.

Tâches

Q :100%