



**INF3405 – Réseaux informatiques
Groupe 01**

Été court 2017

Rapport du Laboratoire 4

Par:

1681547 – Anthony Abboud

1802812 – Kevin Ka Hin Chan

Soumis à :

Fabien Berquez

13 juin 2017

Introduction

Le contexte de ce dernier laboratoire du cours est d'aider un professeur à concevoir et implémenter une application lui permettant d'émettre des sondages à ses étudiants.

Le but de ce TP est donc d'implémenter une application serveur et client (un peu comme le TP3) qui rencontrera toutes les contraintes énoncées à la page 3 des instructions du laboratoire.

Pour l'application serveur, elle devra d'abord saisir quelques paramètres : l'adresse IP de l'ordinateur, le port d'écoute utilisé pour le sondage (situé entre 6000 et 6050) ainsi que la durée du sondage en secondes. Ensuite, elle saisira la question à poser aux élèves qui se connecteront au sondage. Le serveur recevra toutes les réponses des connexions jusqu'à ce que le décompte se termine. De plus, à chaque fois qu'il reçoit une réponse, celle-ci sera affichée à l'écran ainsi que dans un fichier journal.txt.

Du côté de l'application client, elle demandera d'abord à l'élève quel est l'adresse IP du serveur ainsi que le port d'écoute. Après avoir vérifié la validité de ces entrées (tout comme chez le serveur), l'application tentera la connexion avec le serveur. Si c'est un succès, la question du sondage sera transmise à l'élève et il pourra répondre avec sa réponse. Si jamais le sondage est déjà terminé, il recevra le message "Le sondage a expiré". À la fin de son utilisation, le client se déconnectera du serveur.

Ainsi, le but de ce TP est de pouvoir englober tout ce qu'on a appris dans ce cours et de le mettre en pratique en implémentant complètement une application serveur et une application client ainsi que de bien maîtriser les communications.

Présentation du travail

Application Serveur

D'abord, pour l'application serveur, nous avons exécuté les étapes dans l'ordre suivant :

- Saisie des paramètres
- Saisie de la question
- Ouverture du sondage
- Sauvegarde des réponses

Saisie des paramètres

La fonction saisirParametres() permet d'enregistrer l'adresse IP du serveur, son port d'écoute ainsi que la durée du sondage. Elle effectue aussi la validation de ces entrées, comme on peut le voir dans l'image suivante :

```
Parametres du serveur

Entrer l'adresse IP du poste du serveur: a
Entrer l'adresse IP du poste du serveur: 431
Entrer l'adresse IP du poste du serveur: #@$
Entrer l'adresse IP du poste du serveur: 132.23.2
Entrer l'adresse IP du poste du serveur: 132.207.29.115 ← Entrées valides
Entrer le port d'ecoute (entre 6000 et 6050): 1
Entrer le port d'ecoute (entre 6000 et 6050): a
Entrer le port d'ecoute (entre 6000 et 6050): 599
Entrer le port d'ecoute (entre 6000 et 6050): 6051
Entrer le port d'ecoute (entre 6000 et 6050): 6000 ←
Entrer la duree du sondage (en secondes): d
Entrer la duree du sondage (en secondes): 23 ←
```

Saisie de la question

Similairement à la section précédente, la fonction saisirQuestion() enregistre la question du sondage et vérifie qu'elle ne dépasse pas 200 caractères :

```
// Trouver la longueur de la question
int index = 0;
// On cherche ou la question se termine (\0)
for (int i = 0; i < 1001; i++) {
    if (questionTemp[i] == '\0') { index = i; break; }
}
// La position de \0 indique la longueur de la question
if (0 < index && index < 200) {
    for (int i = 0; i < index; i++) {
        question[i] = questionTemp[i];
    }
    toggle = true;
}
else {
    // Si la longueur n'est pas respectee, on recommence..
    cout << endl << "Question trop longue ou trop courte!";
    toggle = false;
}
```

Ouverture du sondage

C'est ici que l'application serveur utilisera les paramètres pour ouvrir le sondage ainsi que de débiter le décompte. Un socket est créé, et on enregistre les informations dans une variable globale à chaque fois qu'on reçoit une réponse d'un élève :

```
if (sd != INVALID_SOCKET) {
    cout << "Connection acceptee De : " <<
        inet_ntoa(sinRemote.sin_addr) << " : " <<
        ntohs(sinRemote.sin_port) << "." <<
        endl;
    informationsAdresse = inet_ntoa(sinRemote.sin_addr);
    informationsPort = ntohs(sinRemote.sin_port);
    DWORD nThreadID;
    CreateThread(0, 0, EchoHandler, (void*)sd, 0, &nThreadID);
}
```

Les informations seront ainsi utilisées dans la fonction EchoHandler afin d'afficher la réponse à l'écran et écrire celle-ci dans un fichier texte.

```
readBytes = recv(sd, readBuffer, 300, 0);
string reponse = readBuffer;
if (readBytes > 0) {
    cout << informationsAdresse << " : " << informationsPort << " - " << reponse << endl;
    sauvegarderReponse(reponse);
    informationsAdresse.clear();
}
```

Lorsque le sondage expire, le string question qui est envoyé change pour :

```
if (temps >= dureeSondage) {  
    toggle = false;  
    strcpy(question, "Le sondage a expire!");  
    cout << "Temps expire, sondage termine." << endl;  
    sd = shutdown(ServerSocket, 2);  
}
```

Et le serveur n'accepte plus de question (puisque l'on peut voir que le socket a été éteint avec la fonction `shutdown()`).

Sauvegarde des réponses

Au tout début du programme, on ouvre un fichier en mode `ios::trunc` et on le referme tout de suite. Cela permet d'effacer les résultats d'un potentiel sondage précédent afin de ne pas mélanger les réponses. Ensuite, lors de la réception des réponses des élèves, on rouvre ce fichier, mais cette fois en mode `ios::app`, afin de garder les réponses à chaque fois qu'on en reçoit une autre.

```
void sauvegarderReponse(string reponse) {  
    ofstream journal;  
    journal.open("journal.txt", ios::app);  
    journal << informationsAdresse << " : " << infor  
    journal.close();  
}  
  
int main(void) {  
  
    // Efface le fichier journal pour stocker les r  
    ofstream resetJournal;  
    resetJournal.open("journal.txt", ios::trunc);  
    resetJournal.close();
```

Application Client

Ensuite, pour l'application client, nous avons exécuté les étapes dans l'ordre suivant :

- Saisie des paramètres
- Validation des paramètres
- Création du socket
- Connexion avec le serveur
- Réponse à la question du sondage

Saisie et validation des paramètres

Tout comme chez le serveur, on demande d'abord à l'élève quel est l'adresse IP du serveur ainsi que son port d'écoute. Ensuite, pour valider ces deux entrées, nous avons implémenté plusieurs fonctions de vérifications :

- `estCaractereException()` : Vérifie si les caractères entrés sont autres que l'intervalle 0 à 9.
- `estPoint()` : Vérifie si le caractère lu est un point.

- `estLongueurValide()` : Vérifie si la longueur de l'adresse IP entrée est une longueur valide, comme par exemple refuser l'entrée 1234.12345667.2.142134213412
- `estFormatIP()` : Fonction utilisant toutes les fonctions énumérées ci-haut afin de vérifier que le format de l'adresse IP est valide.
- `estPortValide()` : Vérifie que le port d'écoute est valide.

Communication avec le serveur

Une fois la connexion avec le serveur réussie, le client utilise `recv()` pour recevoir la question et `send()` pour transmettre sa réponse.

```
//-----
// On va recevoir la question du serveur
iResult = recv(leSocket, motRecu, 199, 0);
if (iResult > 0) {
    printf("Question: ");
    printf(motRecu);
}
else {
    printf("Erreur de reception : %d\n", WSAGetLastError());
}

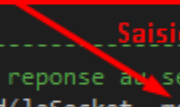
//-----
// Demander à l'utilisateur de répondre à la question
printf("\nReponse a la question (maximum de 300 caracteres): ");
gets_s(motEnvoye);

//-----
// Envoyer la réponse au serveur
iResult = send(leSocket, motEnvoye, 299, 0);
if (iResult == SOCKET_ERROR) {
    printf("Erreur du send: %d\n", WSAGetLastError());
    closesocket(leSocket);
    WSACleanup();
    printf("Appuyez une touche pour finir\n");
    getchar();

    return 1;
}
else { printf("Message reçu! "); }
```

Affichage de la question

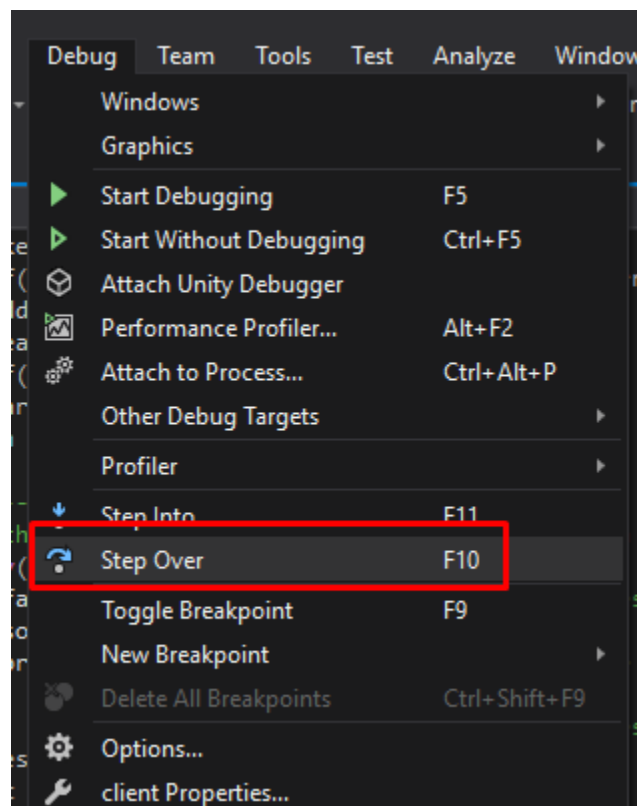
Saisie de la réponse et transmission



Difficultés rencontrées

La première difficulté rencontrée dans ce laboratoire était bien évidemment de maîtriser et bien comprendre l'utilisation des sockets en C++. Pour surmonter cet obstacle, nous avons utilisé une bonne partie du code du TP3 afin de comprendre comme les initialisations se produisaient ainsi que la liste exhaustive d'erreurs de socket et comment le socket permettant de faire des communications.

Ensuite, notre deuxième difficulté était de comprendre les communications entre serveur et client, c'est-à-dire les fonctions `recv()` et `send()` ainsi que les paramètres envoyés. Pour résoudre cette difficulté, nous sommes retourné au TP3 et avons beaucoup utilisé l'outil F10 de Visual Studio :



Cet outil permet d'avancer dans le code ligne par ligne et de bien voir ce qui est transmis à chaque variable. À l'aide de F10, nous avons pu très bien comprendre comment fonctionnait l'entièreté du code.

Notre troisième difficulté rencontrée fut la validation des saisies de paramètres puisqu'il y avait tellement d'exception à analyser et donc il fallait émettre des tests exhaustifs afin de pouvoir les identifier. La solution afin de bien traiter chaque erreur fut évidemment de créer une fonction par exception afin de ne pas se perdre.

Critiques et améliorations

Afin d'améliorer ce laboratoire, nous pensons vraiment que l'ajout de la fonctionnalité de créer une interface pour ce laboratoire serait intéressant. Il sera aussi vraiment intéressant d'apprendre à transmettre d'autres types de données (telles que la voix, une image, une vidéo...).

Conclusion

En conclusion, le laboratoire nous a vraiment aider à comprendre le côté pratique de ce cours. Nous avons appris comment implémenter une application serveur ainsi qu'une application client, pouvant communiquer entre elles. Nous avons aussi appris l'utilisation des sockets, et des transmissions de données en générales.

Nous jugeons que nous aurons maintenant beaucoup plus de facilité à implémenter ce genre de projet dans de futurs travaux ou entreprises.