IUT de BordeauxAnnée 2023-24

Département Informatique S2

**TD R2.05 - Services réseaux**

**Programmation des Sockets TCP en C#**

**Projet Client POP3**

Le but du projet est de développer un Client POP3 en C# en mode graphique. Pour vous aider, un squelette de départ est déjà fourni (partie II sur le projet).

Cependant, **avant de commencer toute programmation**, il est indispensable de bien connaitre le protocole POP3 : la syntaxe des requêtes que le client peut envoyer au serveur, et celle des réponses associées qui reviennent du serveur. C’est l’objectif de la première partie ci-dessous. C’est un rappel et complément de la feuille de TD 1.2.

# (Re)Découverte du protocole POP3

Rappel : POP3 est un protocole pour la consultation d’une boîte à lettre distante : il est exploité par les lecteurs de courriers (Thunderbird, ...) et utilise le port **110** en non sécurisé, et le port **995** en SSL. Voici pour rappel quelques commandes du protocole POP3 :

|  |  |
| --- | --- |
| **Commande** | **Fonction** |
| USER *utilisateur* | Nom du compte de l’utilisateur |
| PASS *mot de passe* | Mot de passe |
| STAT | Donne le nombre de messages non lus et leur taille totale |
| RETR *n* | Récupère le message numéro *n* |
| DELE *n* | Supprime le message numéro |
| LAST | Donne le numéro du dernier message auquel on a accédé |
| LIST *[n]* | Taille de *n*-ème message ou de tous les messages |
| RSET | Annule la suppression de tous les messages |
| TOP *n* *k* | Affiche les entêtes et *k* lignes du message numéro *n* |
| NOOP | Ne fait rien |
| QUIT | Termine la session POP3 |

Voici aussi pour rappel un exemple de scénario qui, après une phase d’authentification, demande le transfert d’un message (réponses du serveur surlignées en bleu) :

S>+OK QPOP (version 2.53) at boite.info.iuta.u

-

bordeaux.fr starting.

C>USER felix

S>+OK Password required for felix.

C>PASS xxxxxxxx

S>+OK felix has 2 messages (2828 octets).

C>STAT

S>+OK 2 2828

C>LIST

S>+OK 2 messages (2828 octets)

S>1 182

3

S>

5

2 100

.

S>

C>RETR 2

S>+OK 1005 octet

s

S>Return

-

Path: <felix@labri.fr>

S>Received:

from smtprt16.wanadoo.fr (smtprt16.wanadoo.fr [193.252.19.183])

Status: R

>

S

O

>

S

S>c

ouco

u

S>

p

f

S>

.

Vous allez dans cet exercice lire le courrier sur le serveur POP3 **pop.laposte.net** de l’utilisateur « **iutinfo** » : Utilisateur : **iutinfo** Mot de passe : **Iutinfo2022!**

**IMPORTANT** : merci de ne PAS SUPPRIMER de mails, c’est-à-dire ne PAS UTILISER la commande DELE de POP3 ! Vous pouvez cependant envoyer des mails si vous le souhaitez (**iutinfo@laposte.net**).

## **Lecture via la commande openssl sous GIT Bash**

* 1- Ouvrez un terminal interpréteur de commandes **Git Bash** et connectez-vous en SSL au serveur de mails POP3 de **laposte.net** avec la commande **openssl** vue lors du TD 1.2 :

**openssl s\_client -connect pop.laposte.net:995**

* 2- Dialoguez avec le serveur en envoyant (saisie puis entrer) une par une les commandes POP3, en vous inspirant du scénario fourni précédemment, pour accéder à la boite mails du compte **iutinfo** (récupérer le nombre de messages, lire des messages, etc…)

## **Analyse fine du protocole POP3 (en vue de la programmation d’un client !)**

Tout dialogue commence par une phase d’authentification (nom et mot de passe). Après une phase d’authentification réussie, le client peut consulter sa boîte à lettres en émettant des requêtes.

Le client envoie des lignes de commande (ou requêtes) et le serveur POP3 répond par une ligne commençant par +OK (ou –ERR en cas de problème).

* 1- Quelles sont les commandes pour lesquelles la réponse du serveur ne contient qu’une seule ligne +OK ?
* USER, PASS, NOOP, STAT, LAST

* 2- Quelles sont les commandes pour lesquelles la réponse du serveur contient plusieurs lignes, une +OK puis les suivantes ?
* RETR, TOP

* 3- Remarquez la règle de protocole suivante : **Toute réponse d’un serveur POP3 qui contient plusieurs**

**lignes est terminée par une ligne contenant un caractère ’.’ tout seul en début de ligne.**

**Et « par malchance », si une ligne de données commence par un caractère ’.’, celui-ci est doublé à l’envoi par le serveur.** Ainsi, à la réception, le client ne peut pas confondre une ligne de données qui ne contenait qu’un point à l’origine (elle est alors codée "**..**") avec la ligne de fin ("**.**"). Pour plus d’informations, voir la RFC !

* 4. Vérifiez la remarque précédente en lisant avec openssl un message de l’utilisateur « **iutinfo** » qui contient des lignes avec un ou des ‘.’ en début de ligne. (Exemple : en principe, le message 3 actuel si personne n’a effacé de messages…)

**Remarques importantes sur la programmation d’un client POP3 :**

Quel que soit le type de client, qu’il soit en mode console ou, comme le vôtre ci-après en mode graphique comme un webmail ou Thunderbird ou autre, le client ne doit pas montrer à l’utilisateur final derrière son écran les ajouts du protocole pour fonctionner.

**Donc en particulier, lors de l’affichage de la liste des mails par exemple (LIST), ou du contenu d’un message (RETR), votre client ne doit pas afficher le caractère ’.’ final ou les caractères ’.’ doublés en début de ligne s’il y en a, vous devez algorithmiquement faire en sorte de ne pas les afficher.**

# Programmation en C# d’un client POP3

**Réalisation du projet :** par équipe de 2 étudiants.

**Remise du Projet en 2 parties : sur Moodle au plus tard mardi 26 mars 2024 à 21h**

1. l’archive du programme produit,
2. un rapport, avec les noms des étudiants de l’équipe, décrivant l’ensemble du travail réalisé.

Le tout fera l’objet d’une note de Projet incluse dans la note de contrôle continu.

## Partie 1 : Découverte – Conseils essentiels – Codage d’une 1ère fonctionnalité (LIST)

**Archive des sources fournies :** récupérez sur Moodle **ClientPOP3Graphique.zip** et désarchivez. Les sources du squelette de départ sont fournies au sein d’une solution Visual Studio en C# dans le répertoire **ClientPOP3Graphique**.

L’application fournie est une ébauche d’un client pour POP3. Elle réalise la connexion au serveur, l’envoi du nom de l’utilisateur et de son mot de passe, et récupère les réponses du serveur.

Elle propose aussi comme exemple les fonctionnalités « STAT » et « QUIT » qui sont déjà programmées.

1.1) Ouvrez la solution avec Visual Studio en cliquant sur **ClientPOP3.sln**.

Consultez la fenêtre Design et les 4 fichiers sources :

* Preferences.cs : regroupe les valeurs paramètres du programme (nom du serveur POP3, port, nom et mot de passe utilisateur) – ***Pas de modification à effectuer ici****.*
* Program.cs : c’est le point d’entrée du programme, – ***Pas de modification à effectuer ici****.*
* Communication.cs : regroupe toutes les fonctions nécessaires aux échanges avec le serveur et la gestion de l’ensemble des fonctionnalités. Ce fichier est bien sûr à modifier dans la suite.

Remarque : pour communiquer en SSL avec le serveur, un stream **ssl** de type **SslStream** est utilisé et les fonctions nécessaires pour ouvrir la connexion SSL ont été ajoutées.

* ClientPOP3.cs (et Design) : regroupe toutes les fonctions nécessaires à la gestion de l’interface utilisateur : affichage dans les listBox, les boutons, etc… et les appels aux fonctionnalités de communication. Ce fichier est bien sûr à modifier dans la suite.

1.2) Exécutez cette application, testez-la, et familiarisez-vous avec le code source fourni. En particulier, il est crucial de comprendre :

* Il y a 2 types d’affichage dans 2 listBox différentes o **listBoxAffichage** (sur fond blanc) : les affichages propres et mis en forme à destination de

l’utilisateur final

o **listBoxVerbose** (sur fond gris) : les affichages « bruts » des envois vers le serveur et de ses réponses : c’est utilisé dans les fonctions fournies de communication avec le serveur **LireLigne** et **EcrireLigne**, à chaque fois qu’une lecture ou écriture est faite sur la socket, l’affichage correspondant est fait dans **listBoxVerbose**. Consultez ces fonctions !

* **Il est donc essentiel d’utiliser, tout le temps et uniquement, ces deux fonctions LireLigne et EcrireLigne pour les lectures des réponses du serveur et les envois vers le serveur**.
* L’organisation de la fenêtre graphique, et la gestion du lien vers le traitement des fonctionnalités de communication, pour pouvoir par la suite, rajouter simplement les nouvelles fonctionnalités.
* La manière dont sont gérés les premiers échanges avec le serveur : USER, PASS et la fonctionnalité STAT, sans ça, vous serez incapable d’écrire les autres fonctionnalités !

1.3) Le bouton de la fonctionnalité **LIST** ne fonctionne pas pour l’instant, car le contenu de la fonction **Communication.List()** est en commentaires. Décommentez ce contenu. Il reste juste à compléter la boucle de traitement des lignes de réponse du serveur fournie.

**Par la suite, vous pourrez vous en inspirer pour les autres fonctionnalités avec réponses sur plusieurs lignes !**

***Indices pour la suite : pour manipuler les string, C# propose de nombreuses propriétés et méthodes comme Length, Substring(), Contains(), Split(), Remove(), etc…, pensez à les utiliser !***

## Partie 2 : Fonctionnalités obligatoires

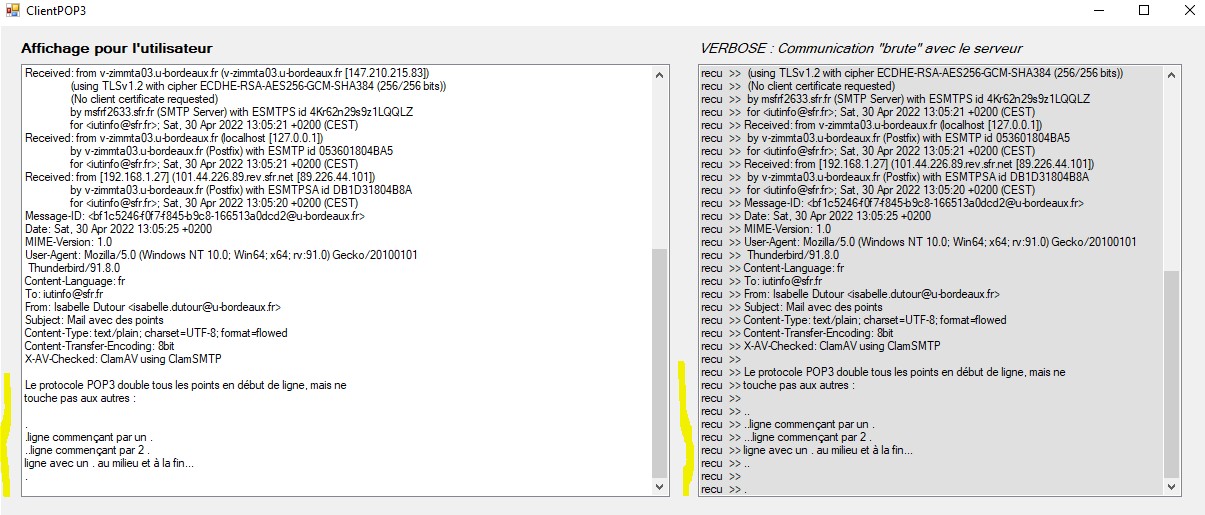
Rajoutez au programme les fonctionnalités suivantes, par exemple en rajoutant des boutons pour chacune d’elles.

2.1) **Afficher le *n*-ème message** : entêtes, contenu, tout compris, sauf bien sûr les points liés au protocole POP3. Vous réaliserez cette fonctionnalité en 3 étapes :

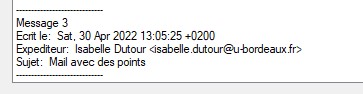
1. Ajoutez par exemple un numericUpDown pour demander la valeur de *n* à l’utilisateur avant qu’il ne clique sur le bouton de la fonctionnalité « Afficher message ».

1. Affichez tout d’abord le message en entier, tel qu’il arrive du serveur, sans vous soucier des points liés au protocole POP3.

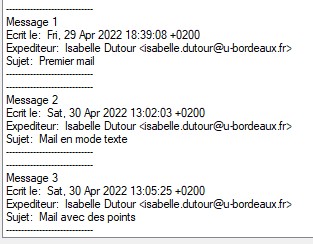
1. Modifiez légèrement votre algorithme d’affichage pour éliminer les points liés au protocole POP3 dans la fenêtre d’affichage pour l’utilisateur (fenêtre de gauche). Voici ci-dessous un exemple de ce que vous devriez obtenir :



* 1. **Afficher uniquement la date, le nom de l’expéditeur et le sujet du *n*-ème message**. Voici ci-dessous un exemple de ce que vous devriez obtenir :



* 1. **Afficher la date, le nom de l’expéditeur et le sujet de tous les messages**. Voici ci-dessous un exemple de ce que vous devriez obtenir :



* 1. **Améliorer l’affichage du *n*-ème message** : plutôt que tout afficher, entête complète + message, trier un peu ce qu'on veut afficher dans l'entête, comme le font nos gestionnaires de mails !

## Partie 3 : Extensions

Imaginez des extensions des fonctionnalités précédentes, ou/et de nouvelles fonctionnalités de votre choix.

Voici une liste (non exhaustive !) de quelques exemples possibles :

* **Ajouter une bannière de connexion** : au lancement du programme, ouvrir une boite de dialogue permettant de se connecter au serveur POP3 voulu (machine, identifiant, mot de passe).

* **Ajouter la déconnexion / reconnexion** : permettre à l’utilisateur de se déconnecter, puis se reconnecter (avec la boite de dialogue) depuis l'application sans devoir la fermer et la relancer.

* **Améliorer globalement l’affichage des messages** : par exemple, avoir une liste des numéros/champs from/subject, et quand on clique dessus ça affiche le contenu du message.

* Etc…