Projet ACII

Un client POP graphique

AFFES Mohamed Amin « 3262731 » - 18 mai 2015

Ce projet est l'implémentation d'un client POP exécutant les requêtes de base de ce protocole en lecture. Un exemple d'une suite de commandes, représentants toutes les requêtes implémentés dans ce projet se trouve dans le fichier « test.txt », se trouvant dans le dossier principal du projet.

Les instructions pour exécuter les tests sont précisés dans le fichier « README » se trouvant dans le dossier principal du projet.

Les deux premières parties du projet sont fonctionnels, je n'ai cependant pas eu le temps de faire la troisième partie. Dans ce rapport je décrirait pour chaque partie faite du projet les grandes lignes de chaque programme permettant d'avoir le résultat attendu. Le code est commenté pour faciliter la compréhension lors de la lecture.

Pensant que le serveur ne devait être fourni, j'ai récupéré celui d'un élève d'ACII pour tester mon programme dessus ne voyant pas la nécessité de la re faire moi même vu qu'il ne faisait pas partie des fichiers demandés dans ce projet. J'ai fourni cependant ce serveur dans mon projet car j'ai lu ce matin votre réponse à la question d'un élève disant que le serveur devait être fourni avec le code ce pourquoi je le fourni et précise ce point.

PARTIE 1

Dans cette partie, une interaction de type « telnet » est proposé, en lançant donc le programme vous pouvez donc écrire sur l'entrée standard une suite de commandes, et lire sur la sortie standard les réponses du serveur.

La fonction effectuant ceci est la fonction « textuel_pop » se trouvant dans le fichier « textuel-pop.c », elle prend en argument un entier représentant la socket permettant la communication avec le serveur. Cette socket est crée dans le programme principal (le fichier « main-pop.c »), utilisant la fonction du fichier « InitConnexion.c » pour initialiser la connexion.

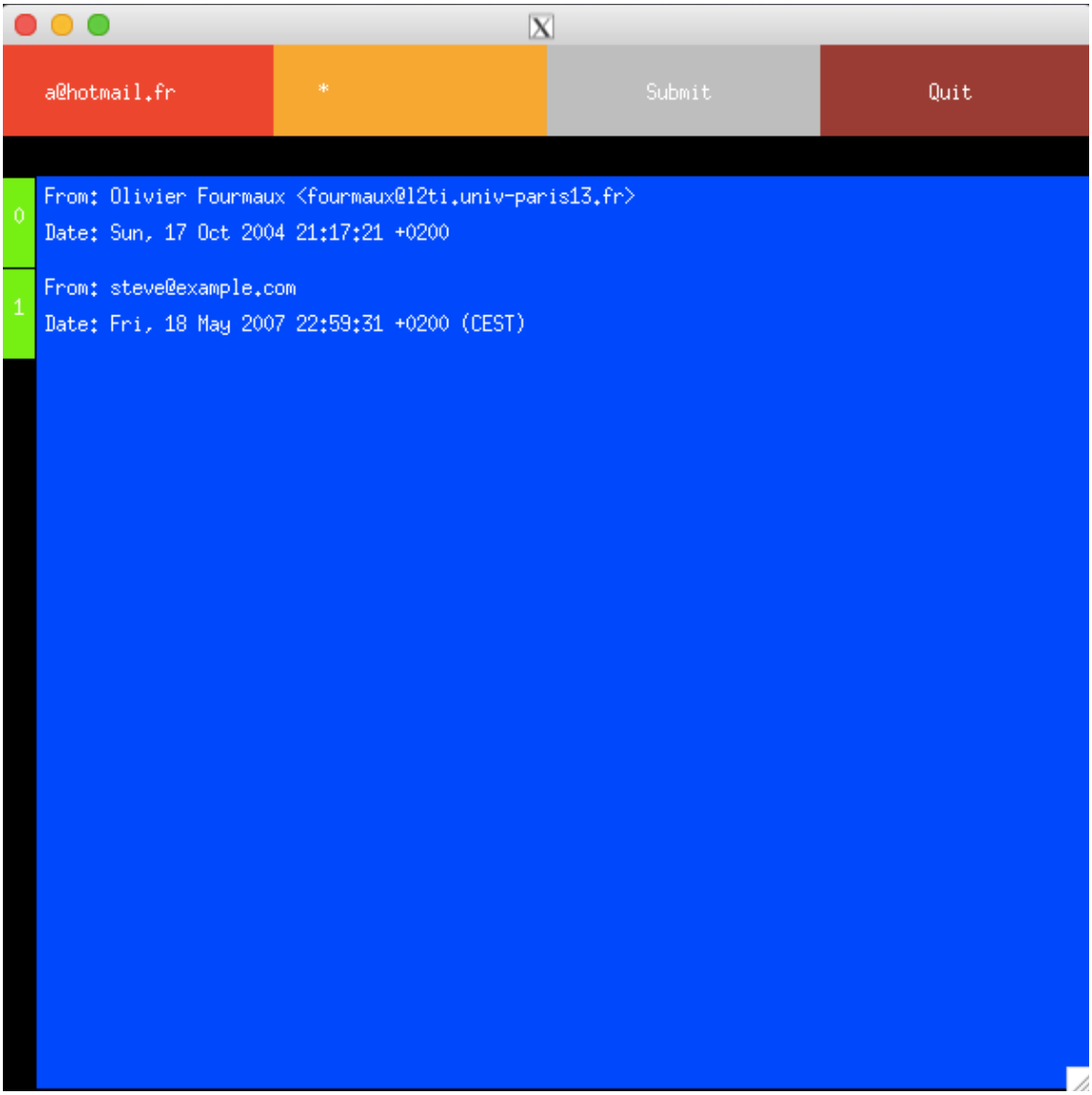
Le programme est une boucle infinie qui attend donc qu'une ligne soit tapé sur l'entrée standard, ensuite il analyse la chaine entrée par l'utilisateur pour voir si la syntaxe de la requête est correcte, et ceci par le biais de tests effectués sur la chaine, en utilisant des expressions régulières définies dans le fichier « peroraison.h ». Si la requête du client est incorrecte la chaine « -ERR » est affichée et le client est prié de re saisir une autre requête. Sinon, la requête demandé est envoyé au serveur, puis sa réponse est affiché.

- Si la requête est une requête « QUIT », la boucle est arrêté et le programme s'arrête.
- Si la requête est une requête « RETR », nous avons trois cas de figures:
  1. Soit le Content-Type est absent, l'intégralité du message est donc sauvé dans un fichier dont le nom est le numéro du message suivi de « `.txt` » .
  2. Soit l'entête est de type MIME simple, l'intégralité du message est donc sauvé dans un fichier dont le nom est le numéro du message suivi de l'extension canonique de ce type.
  3. Soit l'entête est de type MIME `multipart`, on crée alors un répertoire dont le nom est le numéro du message, et son contenu sont les différentes parties du message, selon le nommage ci-dessus.

Le traitement de cette dernière commande est effectué dans la fonction « traitement_RETR » du fichier « textuel-pop.c ».

Nous remarquons à la fin de cette partie que l'utilisation de ce client POP n'est pas du tout intuitif, encore moins pour un utilisateur n'ayant pas un minimum de notions informatique lui permettant l'exécution du programme . Une solution serait donc d'utiliser une interface graphique, ce qui sera présenté dans la partie suivante de ce projet.

PARTIE 2



Cette partie reprend le même principe que la partie précédente, mais en ajoutant une interface graphique facilitant l'interaction au client avec le serveur. Cette interface peux récupérer au maximum 10 messages du serveur.

Nous avons quatre sous fenêtres sur le haut de la fenêtre principale, la première (allant de la gauche) permettant de prendre l'identifiant du l'utilisateur, la suivante le mot de passe, si la chaine dépasse la longueur maximale autorisé c'est à dire **14 caractères**, la chaine est remise à vide. Les deux dernières sous fenêtres sont: la touche Submit permettent la soumission du contenu de l'identifiant et mot de passe de l'utilisateur, et affichant un message d'erreur en cas de l'absence d'un ou plusieurs champs demandés ou

si la réponse du serveur est négative. Puis la touche QUIT la déconnexion de l'utilisateur (s'il est connecté) puis la fermeture du programme.
ATTENTION: Utilisant le code de la touche du clavier tapée lors de la saisie de l'identifiant ou du mot de passe, ce code diffère d'un clavier à un autre. Donc, sur mon clavier (Macbook Air 13 pouces) la touche permettant d'effacer un caractère à la fin de la chaine fonctionne. Ça n'est pas le cas sur certains autre claviers. J'ai essayé d'utiliser la constante « XK_Delete » cependant cela n'a pas fonctionné.

Après la connexion du client, le nombre de messages est récupéré par le biais de la requête « LIST » permettant l'affichage des sous fenêtres contenant le numéro du message, le clic sur l'une de ses fenêtres permet l'envois de la requête « RETR numMessage », la suite est la même que précédemment (création de fichiers ou répertoires selon le content-type). Enfin, une requête « TOP » est envoyé afin d'afficher pour chaque message le contenu des entêtes From et Date dans la dernière sous fenêtre de notre fenêtre principale. La sous fenêtre coloré en noir en dessous des quatre boutons présentés précédemment est la fenêtre affichant les messages d'erreurs en cas de problèmes lors de la saisie (champs vide, identifiant ou mot de passe incorrect ..)

Un exemple d'exécution est présenté dans l'image se trouvant au début de cette section.

Les sous fenêtres sont sensibilisés aux événement les concernants, tous attendent l'événement **expose** permettant l'affichage de chaines de caractères, Toutes les sous fenêtres sauf celle affichant les messages d'erreur et celle affichant les entêtes From et Date sont sensibilisés au clic de la souris. Enfin les deux sous fenêtres permettant de taper l'identifiant et le mot de passe sont sensibilisés à l'événement du clavier.

Les chaines « Submit » et « Quit » sont écrits lors de la réception du premier événement de type **expose**, la chaine représentant l'identifiant de l'utilisateur est modifié à chaque événement du clavier après un premier clic pour sélectionner la fenêtre avant de taper l'identifiant. La chaine représentant le mot de passe est remplacé par des caractères « * » sa longueur étant égale à celle du mot de passe tapé. La récupération du caractère tapé sur la clavier se fait par le biais de la méthode « XLookupString » permettant de récupérer le caractère à partir du **keycode** de la touche tapé.

Une fonction « clean » a été écrite prenant en paramètre une fenêtre et permettant d'effacer le contenu de la fenêtre concernée. Elle est utilisé l'ors de la mise à jour de la chaine caractère écrite sur une fenêtre.

La fonction « ListAndTop » permet l'envoie des requêtes « List » et « Top » afin d'afficher les sous fenêtres affichant les numéros des messages, et les entêtes From et Date des messages.

La fonction « RETR » permet l'envoie de la requête « RETR » pour le message cliqué par l'utilisateur puis effectue les mêmes étapes que pour le mode textuel.

Nous pouvons conclure de ces deux partie que l'interface graphique est un outil très essentiel, permettant l'utilisation d'applications de façon très intuitifs et donc visant un public plus grand. Nous pouvons voir aussi les difficultés rencontrés lorsqu'on code en utilisant les fonction XWindow plutôt que des fonctions de bibliothèques graphiques plus haut niveau. Cela m'a donc permis de voir comment se passe les choses en dessous et donc de mieux comprendre le comportement des interfaces graphiques, et la façon dont certaines fonctions de bibliothèques graphiques sont faites.