Projet : application mail guide

Activite 1

Le but de ce projet va être de créer une application mail. Cette application sera diviser en deux parties : une partie client et une partie serveur. Le client va représenter une boite mail avec une adresse mail, il va envoyer un mail vers une autre adresse mail (une autre exécution du client). Ce mail va être envoyé au serveur qui va l'envoyer vers le bon client.

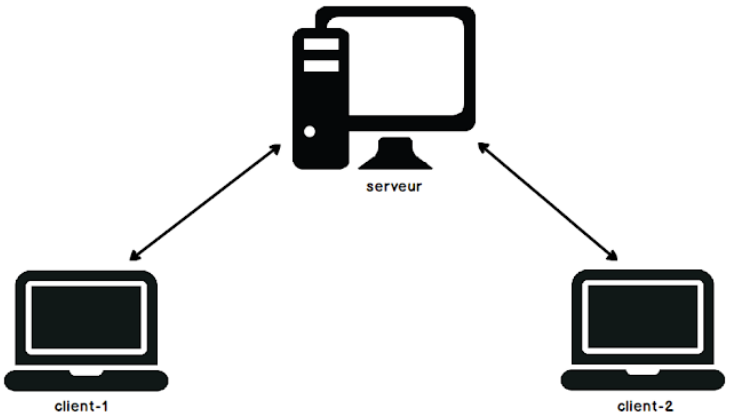

Figure 1: Architecture client serveur

Il y a donc 2 codes, 2 applications a faire. Le premier va être le serveur qui sera exécute une seule fois, et plusieurs clients, potentiellement de plusieurs personnes différentes ou plusieurs exécutions d'un même client pour tester le fonctionnement.

Le développement de cette application se fera en plusieurs étapes (aussi appelé spring), chaque étapes rajoutant des fonctionnalités supplémentaires.

L'application finale fonctionnera selon l'algorigramme 2.

Vous devrez utiliser 2 structures dans ce projet : la structure Membre et la structure Mail. Ces 2 structures sont utilisées dans le serveur. Seul la structure Mail est utilisé dans le client client mais elle est légèrement différente avec celle du serveur (avec globalement le même fonctionnement).

La structure membre est composée de la façon suivante :

- une adresse IP
- un socket
- une adresse Mail

Cette structure permet pour le serveur de définir de manière unique un client et notamment de faire le lien entre son adresse mail et son socket.

La structure Mail est composée de la façon suivante :

- un id qui permet de définir de manière unique un mail
- un receiver (qui sera un membre pour le serveur et une adresse mail pour le client) qui permet de savoir à qui va être envoyé le mail

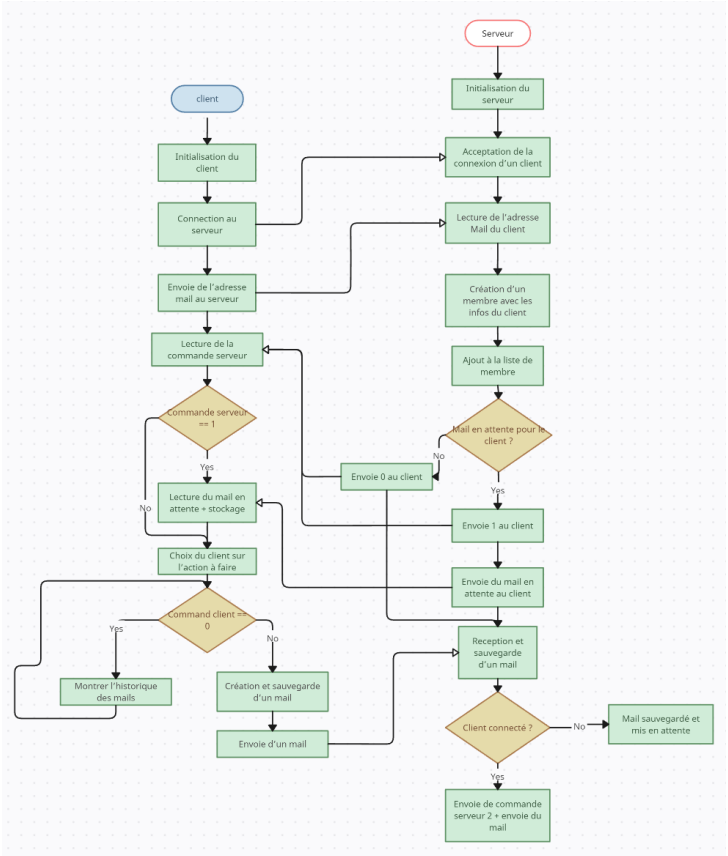

Figure 2: Algorigramme de l'application

- un sender (qui sera un membre pour le serveur et une adresse mail pour le client) qui permet de savoir qui a envoyé le mail
- un message
- un objet de mail
- une date d'envoie (on utilisera le type time_t)
- un état qui est un entier qui définit si le mail a été reçu (1), envoyé mais pas reçu (0) ou -1 si on ne connait pas le receiver.

Le premier spring va consister a réaliser un client serveur basique qui vont juste s'envoyer un message et retourner le même.

Pour le serveur, après la création du socket, je vous conseille de rajouter la ligne de code suivante :

```
if(setsockopt(socket, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) < 0)
    perror("setsockopt(SO_REUSEADDR) failed");
```

Cette ligne de code permettra de reconnecter un même client au serveur après déconnexion.

Vous allez commencer par réaliser un serveur auquel le client pourra se connecter. On ne va pas gérer le multi client pour l'instant.

(a) Réalisez le socket du serveur et effectuez le bind de celui avec la structure adapté.

(b) Utilisez la méthode listen pour finir la configuration de notre serveur et mettez en place l'acceptation de nouvelle connection avec la méthode accept.

Vous allez maintenant réaliser un client qui pourra se connecter au serveur.

(c) Réalisez le socket du client et la structure avec les éléments propre au serveur auquel on veut se connecter.

(d) Connectez vous au serveur avec votre client.

(e) Programmez l'envoie d'un message de bienvenue du serveur au client lors de sa connexion. Affichez le message dans la console pour vérifier la bonne réception du message.

(f) Mettez en place l'envoie d'un message de bonne réception par le client vers le serveur après réception du message de bienvenue. Le serveur se contentera de lire le message reçu.

Activite 2

Le deuxième spring sera focaliser sur la mise en place de membre dans le serveur. L'application restera encore sur un mono client qui se connecte à un serveur.

(a) Modifiez votre code pour permettre à l'utilisateur de rentrer une adresse mail qui sera envoyé au serveur après la connexion à celui-ci.

(b) Vérfiez sur le client que l'adresse mail rentrer respecte bien le format suivant : XXX@bts-ciel.fr.

(c) Mettez en place la structure Membre dans notre serveur. Créez le Membre correspondant à notre client après réception de son adresse mail.

(d) Programmez une liste de Membre qui sera mise à jour à chaque nouveau client rentrant. Cette liste contiendra l'ensemble des clients qui se seront connecté au serveur.

(e) Programmez deux fonctions sendMessage() et readMessage() qui vont envoyez un message au socket donné en paramètre. Ces deux fonctions effectueront aussi la gestion d'erreur correspondante et seront implémenté dans le client et le serveur (ca sera les même fonctions dans le serveur et le client).

Activite 3

Le troisième spring sera focalisé sur la mise en place de la structure Mail. Cette structure permettra de codifier l'envoie d'information entre les différents clients. Néanmoins nous allons encore rester sur une architecture mono client avec serveur.

(a) Mettez en place la structure Mail dans le client et le serveur (différents dans les 2 cas) et réalisez une fonction createMail() pour le client qui permet à l'utilisateur de remplir un mail. La fonction permettra pour l'instant de remplir les champs : receiver, sender, message, object et time.

(b) Programmez une fonction printMail() qui permet d'afficher sur la console un mail donné en paramètre.

(c) Réalisez la fonction sendMail() pour le client et le serveur (identique pour l'instant des 2 côté), permettant d'envoyer un mail à un socket donné en paramètre. Ne prenait pas encore en compte les 2 champs ID et état. Vous pouvez regarder la fonction strftime() pour gérer le temps.

(d) Réalisez la fonction readMail() pour le client, permettant de lire un mail un mail à un socket donné en paramètre. Vous pouvez regarder la fonction strptime() pour la gestion du temps. Attention les messages arrivent dans le même ordre qu'ils sont envoyés.

(e) Complétez les fonctions findMemberByAdress() et findMemberBySocket() du serveur pour trouver un membre dans notre liste de membre en fonction d'une adresse ou d'un socket donné en paramètre.

(f) Réalisez la fonction readMail() pour le serveur, vous pouvez vous inspirer de la méthode du client et utiliser les 2 fonctions créer dans la question précédente.

(g) Un problème peut survenir lors de l'envoie de nombreux messages et la réception de ceux ci. Le mieux est d'envoyer un seul message contenant l'ensemble de nos informations, puis de découper le message à la réception. Vous allez modifier les fonctions sendMail() pour qu'elle envoie un seul message constitué de toutes les informations avec un séparateur (on peut utiliser le caractère |) entre chaque informations (ex : adresse | ...). Vous pouvez utilisez la fonction sprintfs ou strcat.

(h) Modifiez la fonction readMail() pour qu'elle réceptionne le message avec les séparateurs, et sépare les diverses informations pour reconstituer un mail. Vous pouvez utilisez la fonction strtok.

(i) Mettez en place les différentes fonctions précédentes pour que le client envoie un mail au serveur avec lui même comme destinataire.

(j) Programmez deux fonctions readInt() et sendInt() qui permettent d'envoyer et recevoir un int par le biais d'une connexion client/serveur.

(k) Mettez en place le principe d'id sur le serveur. Le premier mail reçu aura pour id 0, le deuxième 1, ... Les clients ne donnent pas d'id a leur mail lors de la création, c'est le serveur qui calcule l'id et le renvoie au client lors de la réception.

(l) Programmez sur le serveur une liste de mail en attente. Cette liste sera remplie lorsque le serveur doit envoyer un mail à un client non connecté.

(m) Modifiez la fonction readMail() du serveur pour envoyer la valeur d'état du mail (voir intro) en fonction de si le client est connecté ou non.

(n) Envoyez un nombre au client lors de sa connexion pour signifier si il a des messages en attente ou non (0 si non, 1 si oui). Modifiez le code du client et du serveur en conséquence.

(o) Permettez au serveur d'envoyer un mail non envoyé au client désiré lors de sa connexion. N'oubliez pas d'enlever le mail de la liste après l'envoie.

Activite 4

Le quatrième spring sera dédie a la gestion du multi client. L'approche de cette partie est différente, vous allez avoir un code fournit que vous allez devoir adapter a votre code.

(a) Récupérer le code codeServerMultiClient.c et lisez le pour comprendre le fonctionnement, faites des recherches internets pour comprendre les différentes lignes de code. Vous pouvez remarquer que le code est séparé en 2 parties, une partie qui gère la connexion d'un client et une autre qui gère les envoie de message des clients.

(b) Reprenez le code pour l'ajouter a votre code.

(c) Tester votre code en lancant 2 clients avec 2 adresses mail différentes.

Activite 5

Le cinquième spring sera focalisé sur la sauvegarde des mails sur les différents clients. Le but va être de pouvoir avoir à tout instant les mails reçus par le client, même si on a fermé ou ré ouvert le client entre temps. Tous les mails seront stockés sur le même fichier (saveMail.txt) et ce fichier sera constitué de la façon suivante :

- l'id sur la première ligne
- le mail de l'envoyeur sur la deuxième
- le mail du réceptionner sur la troisième ligne
- l'objet du mail sur la quatrième ligne
- le contenu du message sur la cinquième ligne
- finalement l'heure et la date de l'envoie du message
- et une ligne vide pour faire séparation avec le mail suivant

(a) Créez une fonction pour sauvegarder un mail dans notre client dans notre fichier saveMail.txt

(b) Mettez en place une vérification pour éviter la sauvegarde de 2 mails identiques sur notre fichier, vous pouvez utilisez l'id des mails.

(c) Réalisez une fonction qui prends une liste de mail vide, et la remplie avec l'ensemble des mails sauvegarder sur notre fichier. Le but de cette fonction va être de récupérer tous les mails reçues par notre client lorsqu'on le démarre.

(d) Utilisez les méthodes ainsi créé pour mettre en place la sauvegarde des mails sur notre client.

Activite 6

Le but du sixième spring est de mettre en place le menu dans notre client. Le menu va permettre de donner au client le choix entre plusieurs actions. Ce menu permettra de visualiser un historique de l'ensemble des mails reçu précédemment, de se mettre en mode attente d'un mail entrant, ou alors d'envoyer un mail (et créer). Une communication avec le serveur va être nécessaire pour savoir si le client est en attente ou en rédaction de mail.

(a) Mettez en place un menu qui permet pour l'instant d'accéder à un espace de rédaction de mail et d'envoi de mail .

(b) Complétez le menu en permettant au client d'afficher un historique de tous les mails qu'il a reçu.

(c) Mettez en place une notion d'état pour le client précisant l'action qu'il est en train d'entreprendre au serveur. Le client doit envoyer 0 si il est en train de rédiger un mail, ou 1 si il est dans l'historique ou en attente de mail (on peut recevoir un mail lorsque l'on est dans l'historique).

(d) Finalisez le menu en rajoutant un menu d'attente pour la réception du mail. On doit pouvoir sortir de ce menu pour aller vers les autres menues (et donc envoyer une commande vers le serveur).