

Compte Rendu Projet Réseau

L3 IUP GMI d'Avignon

Exercice 1

Le premier exercice est divisé en deux parties, le client et le serveur. Le serveur possède son main dans la classe Serveur et prend un argument, le numéro de port d'écoute. La classe Serveur crée une instance de la classe ServeurDialogue par client qui se connecte. ServeurDialogue gère toutes les communications avec son client attaché. Juste après sa création, il se transforme en thread pour que les autres instances de ServeurDialogue puissent travailler en même temps.

Le client lui, possède son main dans la classe client2 et prend un argument, le numéro de port d'écoute du serveur. La classe client2 crée une instance de ClientDialogue par client qui se connecte. ClientDialogue gère la réception et le traitement des messages du serveur. Juste après sa création, il se transforme en thread pour que l'instance de Client2 puisse gérer en parallèle la saisie des messages par l'utilisateur, le formatage et l'envoi au serveur des messages.

Format des messages :

- Authentification : login: LOGIN_UTILISATEUR
- message : message : MESSAGE

Vérification :

- Le numéro de port donné en argument doit être un entier compris entre 1024 et 65535
- Vérification du format action : valeur

BONUS :

Réalisation d'une interface graphique sous forme d'applet web.

Exercice 2

Le deuxième exercice ne comporte désormais plus qu'une seule partie, un client/serveur. Le programme possède son main dans la classe Serveur et prend un argument, le numéro de port d'écoute. Au lancement du programme on demande le nombre de serveur déjà démarré ainsi que leur adresse IP et port d'écoute associé. Le programme se connecte ensuite directement au autre serveurs déjà démarré, s'il y en a. La classe Serveur crée une instance de la classe ServeurDialogue par client qui se connecte. ServeurDialogue gère la réception et le traitement des données avec son client attitré. Juste après sa création, il se transforme en thread pour que les autres instances de ServeurDialogue puissent travailler en même temps. La classe Serveur crée une fois les socket créées une instance de ClientDialogue qui se transforme en thread qui puisse gérer en parallèle la saisie des messages par l'utilisateur, le formatage et l'envoi à tous les clients des messages via la fonction envoyerServeur de la classe ServeurDialogue.

Format des messages :

- Authentification : `login: LOGIN_UTILISATEUR`
- message : `login : message > MESSAGE`
- Déconnexion : `general : LOGIN est parti du chat`

Vérification :

- Le numéro de port donné en argument doit être un entier compris entre 1024 et 65535
- Vérification du format `action : valeur`

Exercice 3

Le troisième exercice se compose de deux parties, un serveur d'enregistrement et des clients. Le serveur d'enregistrement possède son main dans la classe `ServeurM` et prend un argument, le numéro de port d'écoute. Le serveur d'enregistrement attend les demandes de requêtes des clients. La classe `ServeurM` crée une instance de la classe `ServeurDialogueM` par client qui se connecte, elle gère aussi une `ArrayList` avec les caractéristiques de chaque clients. `ServeurDialogueM` gère la réception et le traitement des données concernant les connexions/déconnexions des clients avec son client attiré. Juste après sa création, il se transforme en thread pour que les autres instances de `ServeurDialogueM` puissent travailler en même temps.

Le client possède son main dans la classe `Serveur` et prend quatre arguments :

- L'adresse IP du serveur d'enregistrement
- Le numéro de port du serveur d'enregistrement
- L'adresse IP du client
- Le numéro de port du client

Le serveur demande dès le lancement un nom d'utilisateur pour enregistrer le client auprès du serveur. Il crée ensuite une instance de `ClientDialogue` (un thread) qui gère les nouvelles connexions/déconnexions des clients. Cette aussi `ClientDialogue` qui se charge d'établir/de couper les connexions avec les autres clients. La classe serveur crée ensuite une instance de `ShellDialogue` qui se transforme en thread qui puisse gérer en parallèle la saisie des messages par l'utilisateur, le formatage et l'envoi à tous les clients des messages via la fonction `envoyer` de la classe `Serveur`. Pour finir la classe `Serveur` crée ensuite une socket d'écoute et attend des demandes de connexions de la part d'autres clients. Pour chaque demande de connexion, le serveur instancie et lance un thread `ServeurDialogue` qui gère la réception et le traitement des messages envoyés entre les clients du chat.

Format des messages :

- Authentification (serveur) : `login: IP > port`
- Message : `login : message > MESSAGE`
- Déconnexion (clients): `login : message > FIN`
- Déconnexion (serveur): `deco-login`

Schéma de fonctionnement de l'exécice 3

Légende :

- 1 ClientDialoge envoie / recoie les informations sur les conneixons / déconneixons
- 2 ServeurDialogueM envoie les info au clients sur les connexions / déconneixons
- 3 ShellDialoge gère la saisie et l'envoi des messages par le client
- 4 ServeurDialogue établit / coupe les connexions avec les autre clients (toutes les connexions ne sont pas présentes par souci de lisibilité)

