Architecture micro-service

TP 3 - Serveur TCP/IP simple - b. Analyse

Philippe Roussille

3iL INGENIEURS
</les sciences informatiques>

1 À cod- ah non, c'est déjà fait ! Analyse d'un serveur existant – IRC CanaDuck (code fourni)

Exceptionnellement, le code est déjà fait… mais pas encore compris !

Cette fois-ci, ce n'est pas à vous d'écrire le serveur : **Ginette et Roger l'ont déjà fait à votre place**. Le fichier `final.py` contient une version complète (mais brute) du serveur IRC de CanaDuck. Vous allez devoir **le lire, l'analyser, le comprendre et proposer des évolutions**.

1.1 Ce que fait déjà ce serveur

Le serveur accepte plusieurs connexions TCP (clients fournis) et gère les commandes suivantes :

Commande	Effet
`/nick <pseudo>`	Choisir un pseudo (obligatoire)
`/join <canal>`	Rejoindre ou créer un canal
`/msg <texte>`	Envoyer un message dans le canal courant
`/read`	(Message d'info, tout est en direct)
`/log`	Afficher les 10 dernières lignes du journal
`/alert <texte>`	Envoi d'une alerte globale (admin/moderator)
`/quit`	Déconnexion propre

Fonctionnalités techniques :

— Utilisation de `socketserver` et de threads pour gérer plusieurs clients
— Mémorisation de l'état (`etat_serveur.json`) et journalisation (`serveur.log`)
— Diffusion des messages à tous les utilisateurs d'un canal
— Gestion concurrente protégée par un verrou (`threading.Lock`)

2 Ce que vous devez faire maintenant

2.1 Analyse du code

Répondez aux questions suivantes **sans modifier encore le code** :

1. **Qui traite les commandes ?**
   - Quelle fonction interprète `/msg`, `/join`, etc. ?
   - Qui accède à la mémoire partagée `etat_serveur` ?
2. **Où sont stockées les infos ?**
   - Où est enregistré le canal courant d'un utilisateur ?
   - Où sont les flux de sortie (`wfile`) associés à chaque client ?
3. **Qui peut planter ?**
   - Que se passe-t-il si un client quitte sans envoyer `/quit` ?
   - Qu'arrive-t-il si un `write()` échoue ? Est-ce détecté ?
   - Est-ce qu'un canal vide est supprimé ?

2.2 À produire

2.2.1 Un schéma d'architecture fonctionnelle

- Identifiez les composants (serveur, handler, état, log, etc.)
- Reliez-les avec des flèches pour représenter les interactions
- Notez où ont lieu les accès partagés, les verrous, et les éventuels risques

2.2.2 (en option pour ceux qui vont plus vite) Une fiche protocole

- Liste des commandes
- Syntaxe exacte
- Exemples d'utilisation
- Réponse typique du serveur

3 Ce que vous pourriez faire ensuite si vous avez du temps (à coder)

Idée de commande	Objectif
`/list`	Lister tous les canaux actifs
`/users` ou `/who`	Voir les utilisateurs connectés
`/part`	Quitter un canal
`/whisper`	Envoyer un message privé à un utilisateur
`/topic`	Définir ou afficher le sujet d'un canal
`/away [msg]`	Se déclarer momentanément absent

Bonne analyse ! Posez des questions, testez à deux, notez vos idées : ce TP est là pour penser l'architecture autant que pour coder.