Architecture micro-service

TP 3 - Serveur TCP/IP simple - a. Contexte

Philippe Roussille

3iL INGENIEURS
</les sciences informatiques>

1 Présentation du cadre

> *« Vous avez vu ? Maintenant chaque canard a son propre ordi ! » — Roger, chef d'équipe*

Nous sommes au **milieu des années 1990**. CanaDuck, après avoir expérimenté les systèmes monolithiques (TP1) puis centralisés (TP2), investit enfin dans un **véritable réseau TCP/IP local**. Chaque salarié dispose désormais d'un poste de travail individuel, capable de se connecter à un serveur via le réseau.

L'ancien programme IRC centralisé par fichiers ne suffit plus. L'entreprise veut :

- un **serveur IRC** capable de dialoguer avec plusieurs clients simultanément,
- des **messages échangés en direct**, sans fichier intermédiaire,
- une **gestion claire des canaux** et des utilisateurs connectés,
- une **structure modulaire** permettant de faire évoluer le système facilement.

Ce TP marque un **changement d'architecture majeur** : vous entrez dans le monde du **client-serveur réel**, avec une communication réseau (TCP), des **threads pour gérer les clients**, et des **structures d'état mémoire partagées**.

Vous allez écrire le **serveur**, pendant que les clients (déjà fournis) se connectent, envoient des commandes, et reçoivent les réponses au fil de l'eau.

Mais surtout, ce TP est l'occasion de **réfléchir à l'architecture du système** que vous concevez : quelles sont ses briques principales ? Comment les modules communiquent-ils ? Que faudra-t-il changer si on ajoute un service REST, une base de données, ou un système de modération ?

2 Réflexion liminaire

Prenez un moment pour poser les enjeux structurels et conceptuels de ce que vous allez coder.

2.1 Cohérence concurrente et synchronisation

— Quels types de **problèmes de concurrence** peuvent apparaître dans ce système multi-clients ?
— Que peut-il arriver si deux clients rejoignent ou quittent un canal en même temps ?
— Votre système est-il vulnérable aux **incohérences d'état** ou aux conditions de course ? Comment s'en prémunir ?

2.2 Modularité et séparation des responsabilités

— Quelles sont les **grandes responsabilités fonctionnelles** de votre application serveur (gestion client, traitement commande, envoi message, logs, persistance…) ?
— Peut-on tracer une **frontière claire** entre logique métier et logique d'entrée/sortie réseau ?
— En cas d'erreur dans une commande, quelle couche doit réagir ?

2.3 Scalabilité et capacité à évoluer

— Si vous deviez **ajouter une nouvelle commande** (ex : `/topic`, `/invite`, `/ban`), quelle partie du système est concernée ?
— Que faudrait-il pour que ce serveur fonctionne à grande échelle (plusieurs centaines de clients) ?
— Quelles limitations structurelles du code actuel empêchent une montée en charge ?

2.4 Portabilité de l'architecture

— Ce serveur TCP pourrait-il être **adapté en serveur HTTP** ? Quelles parties seraient conservées, quelles parties changeraient ?
— Dans une perspective micro-services, quels modules seraient **candidats naturels** pour devenir des services indépendants ?
— Est-il envisageable de découpler la gestion des utilisateurs de celle des canaux ? Comment ?

2.5 Fiabilité, tolérance aux erreurs, robustesse

— Le serveur sait-il détecter une **déconnexion brutale** d'un client ? Peut-il s'en remettre ?
— Si un message ne peut pas être livré à un client (socket cassée), le système le détecte-t-il ?
— Peut-on **garantir une livraison** ou au moins une trace fiable de ce qui a été tenté/envoyé ?

2.6 Protocole : structuration et évolutivité

— Quelles sont les **règles implicites** du protocole que vous utilisez ? Une ligne = une commande, avec un préfixe (`/msg`, `/join`, etc.) et éventuellement des arguments : est-ce un protocole explicite, documenté, formalisé ?

— Le protocole est-il **robuste** ? Que se passe-t-il si un utilisateur envoie `/msg` sans texte ? Ou un `/join` avec un nom de canal invalide ?
— Peut-on imaginer une **spécification formelle** de ce protocole ? Un mini-ABNF, une doc à destination des développeurs de client ?
— Quelle serait la **différence structurelle** entre ce protocole et un protocole REST ou HTTP ?