Projet Messagerie C - Livrable 2

Wassim Ben Nacef - Nabil Bennacer - Max Chateau

Polytech Montpellier - IG 3

1 Avancement du projet

1.1 Mise en place des commandes

Le système de commandes implémenté est tel qu'indiqué dans le sujet :

- **`@connect <pseudo> <password>`** : Permet à un utilisateur de se connecter avec ses identifiants. Si l'utilisateur n'existe pas, il est automatiquement enregistré avec le mot de passe fourni.
- **`@disconnect`** : Déconnecte l'utilisateur actuel et libère sa session.
- **`@register <pseudo> <password>`** : Enregistre explicitement un nouvel utilisateur dans le système.
- **`@msg <destinataire> <message>`** : Permet d'envoyer un message privé à un utilisateur connecté. Le message est acheminé uniquement vers le destinataire spécifié.
- **`@ping`** : Vérifie la connexion au serveur et renvoie le statut de l'utilisateur (connecté ou non).
- **`@help`** : Affiche l'aide avec la liste des commandes disponibles.
- **`@credits`** : Affiche les informations sur l'équipe de développement du projet.
- **`@shutdown`** : Commande administrative permettant l'arrêt propre du serveur avec notification à tous les utilisateurs connectés.
- **`@upload (ou @download) <fichier>`** : Gestion des fichiers (fonctionnement detaille dans la derniere partie)

L'implémentation repose sur la fonction `getCommandType()` qui analyse le préfixe du message. Chaque commande est ensuite traitée par une fonction dédiée qui vérifie les permissions de l'utilisateur et exécute l'action correspondante.

1.2 Gestion des utilisateurs

Nous avons développé le système de gestion des utilisateurs avec les fonctionnalités suivantes :

- **Authentification** : Les identifiants utilisateur sont vérifiés via un fichier CSV (`users.csv`) qui stocke les couples pseudo/mot de passe.
- **Enregistrement automatique** : Lors d'une première connexion, si l'utilisateur n'existe pas, il est automatiquement enregistré dans le système.
- **Gestion des sessions** : Le serveur maintient une liste des utilisateurs actifs et de leurs informations de connexion (IP, port) dans un tableau `activeUsers`.
- **Détection de multiples connexions** : Le système empêche un même utilisateur de se connecter simultanément depuis plusieurs clients différents.
- **Gestion des droits administrateurs** : Le premier utilisateur à se connecter au serveur reçoit automatiquement des privilèges d'administration, lui permettant notamment d'utiliser la commande `@shutdown`.
- **Structure de données optimisée** : Chaque utilisateur est représenté par une structure `User` contenant :
  - Pseudo (limité à `PSEUDO_MAX` caractères)
  - Mot de passe (stocké en clair pour ce prototype)
  - Adresse IP et port du client
  - État de connexion (connecté/déconnecté)
  - Statut d'administrateur

1.3 Gestion des fichiers

On a implementé un système de transfert de fichiers qui s'appuie sur le protocole TCP pour garantir l'intégrité des données transmises. Contrairement aux messages texte qui utilisent UDP, le transfert de fichiers nécessite une connexion fiable et une confirmation de réception pour chaque partie du fichier, de plus il s'agit d'opérations effectués moins régulièrement que les envois de messages.

- **Architecture double protocole** : Nous utilisons UDP pour la messagerie instantanée et TCP pour le transfert de fichiers, offrant ainsi un bon équilibre entre performance et fiabilité selon le type de données à transmettre.
- **Commandes spécifiques** : Le transfert de fichiers est initié par des commandes dédiées :
  - `@upload <fichier>` : Permet à un utilisateur d'envoyer un fichier au serveur.
  - `@download <fichier>` : Permet à un utilisateur de télécharger un fichier stocké sur le serveur.
- **Mécanisme en deux phases** :
  - Phase 1 : L'utilisateur envoie une requête de transfert via le canal UDP normal.
  - Phase 2 : Le serveur répond par un message de type `TCP:<OPERATION>:<fichier>:<ip>` qui déclenche l'établissement d'une connexion TCP dédiée.
- **Gestion des flux de données** :
  - Envoi par blocs de taille définie (`BUFFER_MAX`)
  - Accusés de réception pour chaque bloc transmis
  - Contrôle des erreurs à chaque étape du processus
- **Sécurité et fiabilité** :
  - Vérification de l'état des fichiers avant transmission
  - Gestion des erreurs à chaque étape du processus
  - On considère qu'un seul utilisateur à la fois peut envoyer ou recevoir un fichier (pour le moment)