Q1 : Quelle est la fonction principale du fichier `main.cpp` ?

R : La fonction principale de `main.cpp` est de démarrer le serveur IRC, écouter les connexions entrantes, et gérer la boucle principale d'événements. Il utilise `poll()` pour surveiller les sockets, accepte les nouvelles connexions et lit les données des clients existants, et gère les commandes reçues des clients.

Q2 : Quelle classe est définie dans `client.hpp` et `client.cpp` ?

R : Les fichiers `client.hpp` et `client.cpp` définissent la classe `Client`, qui représente un utilisateur connecté au serveur IRC. Cette classe gère les informations sur le client, telles que le socket, le pseudonyme, l'authentification, et les canaux auxquels le client est connecté.

Q3 : Quelles sont les principales responsabilités de la classe `Client` ?

R : La classe `Client` est responsable de la gestion des informations sur le client, telles que le socket, le pseudonyme, l'authentification, et les canaux auxquels le client est connecté. Elle implémente également des méthodes pour gérer les commandes client et les interactions avec le serveur.

Q4 : Que représente la classe `Channel` définie dans `channel.hpp` et `channel.cpp` ?

R : La classe `Channel` représente un canal de discussion sur le serveur IRC. Elle gère les informations sur le canal, telles que les utilisateurs, les opérateurs, les modes de canal, et le sujet. Elle implémente des méthodes pour ajouter/retirer des utilisateurs, diffuser des messages, et gérer les modes de canal.

Q5 : Quelles fonctions utilitaires sont fournies dans `utils.hpp` et `utils.cpp` ?

R : Les fichiers `utils.hpp` et `utils.cpp` fournissent des fonctions utilitaires pour le serveur IRC, telles que l'envoi de messages aux clients (`sendToClient`), la manipulation des chaînes de caractères (`trim`, `ltrim`, `rtrim`), et la vérification de caractères invalides dans les pseudonymes (`containsInvalidCharacters`).

Q6 : Quelles commandes IRC sont gérées par les fichiers `irc_commands.hpp` et `irc_commands.cpp` ?

R : Les fichiers `irc_commands.hpp` et `irc_commands.cpp` implémentent les fonctions de traitement des commandes IRC, telles que `PING`, `PONG`, `JOIN`, `PRIVMSG`, `MODE`, `KICK`, `INVITE`, `TOPIC`, `LIST`, `NAMES`, et `WHOIS`. Ils vérifient les permissions des utilisateurs et appliquent les changements nécessaires aux canaux et aux utilisateurs en fonction des commandes reçues.

Q7 : Comment le serveur gère-t-il les connexions multiples et simultanées ?

R : Le serveur gère les connexions multiples et simultanées en utilisant la fonction `poll()` dans `main.cpp`, qui surveille les sockets pour les événements d'entrée/sortie. Cela permet au serveur de traiter les demandes de plusieurs clients en même temps sans se bloquer.

Q8 : Que se passe-t-il lorsque `poll()` détecte une nouvelle connexion entrante ?

R : Lorsque `poll()` détecte une nouvelle connexion entrante, la fonction `accept()` est appelée pour accepter la connexion. Un nouvel objet `Client` est créé pour représenter le nouvel utilisateur, et le socket du client est ajouté à la liste des descripteurs surveillés par `poll()`.

Q9 : Comment le serveur gère-t-il les commandes partielles envoyées par les clients ?

R : Le serveur gère les commandes partielles en stockant les fragments de commandes dans un buffer de commande associé à chaque client. Lorsqu'une ligne complète est reçue (délimitée par un saut de ligne), elle est traitée comme une commande complète.

Q10 : Quelle est la fonction de `fcntl(fd, F_SETFL, O_NONBLOCK);` dans le code du serveur ?

R : La fonction `fcntl(fd, F_SETFL, O_NONBLOCK);` configure le descripteur de fichier (socket) pour qu'il fonctionne en mode non-bloquant. Cela signifie que les appels d'entrée/sortie sur ce descripteur ne bloqueront pas le processus si l'opération ne peut pas être complétée immédiatement.

Q11 : Comment le serveur vérifie-t-il si un utilisateur est autorisé à exécuter une commande spécifique sur un canal ?

R : Le serveur vérifie si un utilisateur est autorisé à exécuter une commande spécifique sur un canal en utilisant les méthodes de la classe `Channel`, telles que `isOperator()` pour vérifier si l'utilisateur est un opérateur du canal, et en comparant le pseudonyme de l'utilisateur avec les permissions requises pour la commande.

Q12 : Que se passe-t-il lorsqu'un utilisateur envoie la commande `JOIN <channel>` ?

R : Lorsque un utilisateur envoie la commande `JOIN <channel>`, le serveur ajoute l'utilisateur au canal spécifié (créant le canal s'il n'existe pas déjà) en utilisant la méthode `addUser()` de la classe `Channel`. L'utilisateur est également informé des autres utilisateurs présents dans le canal et du sujet du canal.

Q13 : Comment le serveur gère-t-il les déconnexions inattendues des clients ?

R : Le serveur gère les déconnexions inattendues en détectant la fermeture du socket client (lorsque `read()` retourne 0 ou une erreur). Le socket du client est alors fermé et supprimé de la liste des descripteurs surveillés, et l'objet `Client` associé est supprimé.

Q14 : Comment le serveur diffuse-t-il un message à tous les utilisateurs d'un canal ?

R : Le serveur diffuse un message à tous les utilisateurs d'un canal en utilisant la méthode `broadcast()` de la classe `Channel`. Cette méthode envoie le message à chaque utilisateur du canal en utilisant leur socket respectif.

Q15 : Quelle est la structure de données utilisée pour stocker les clients et les canaux dans le serveur ?

R : Le serveur utilise des `std::map` pour stocker les clients (`std::map<int, Client> clients`) et les canaux (`std::map<std::string, Channel> channels`). Les clés sont respectivement le descripteur de fichier du socket client et le nom du canal.

Q16 : Comment le serveur gère-t-il les invitations des utilisateurs à un canal ?

R : Le serveur gère les invitations des utilisateurs à un canal en utilisant la commande `INVITE`. Lorsqu'un utilisateur envoie cette commande, l'utilisateur invité est ajouté à la liste des invités du canal et reçoit une notification d'invitation.

Q17 : Comment un utilisateur devient-il opérateur d'un canal ?

R : Un utilisateur devient opérateur d'un canal lorsque la commande `MODE <channel> +o <nickname>` est envoyée par un autre opérateur du canal ou par un administrateur. La méthode `addOperator()` de la classe `Channel` est utilisée pour ajouter l'utilisateur à la liste des opérateurs du canal.

Q18 : Comment le serveur gère-t-il les sujets de canal ?

R : Le serveur gère les sujets de canal en utilisant la commande `TOPIC`. Lorsqu'un utilisateur envoie cette commande, le sujet du canal est mis à jour en utilisant la méthode `setTopic()` de la classe `Channel`. Si le canal est protégé par sujet, seule une personne ayant les permissions appropriées peut changer le sujet.

Q19 : Que se passe-t-il lorsqu'un utilisateur envoie la commande `PART <channel>` ?

R : Lorsqu'un utilisateur envoie la commande `PART <channel>`, il quitte le canal spécifié. La méthode `removeUser()` de la classe `Channel` est utilisée pour supprimer l'utilisateur du canal, et l'utilisateur est retiré de la liste des canaux auxquels il est connecté.

Q20 : Comment le serveur traite-t-il les commandes `PING` et `PONG` ?

R : Le serveur traite les commandes `PING` en répondant avec une commande `PONG` contenant le même message que celui reçu. La commande `PONG` est utilisée pour mettre à jour le temps de la dernière réponse `PONG` reçue par le client, ce qui aide à maintenir la connexion active.

Q21 : Comment le serveur gère-t-il les messages privés entre utilisateurs ?

R : Le serveur gère les messages privés entre utilisateurs en utilisant la commande `PRIVMSG`. Lorsqu'un utilisateur envoie cette commande avec le pseudonyme d'un autre utilisateur, le message est acheminé directement au destinataire. Si le destinataire est un canal, le message est diffusé à tous les membres du canal.

Q22 : Qu'est-ce qu'une commande `MODE` et comment est-elle utilisée ?

R : La commande `MODE` est utilisée pour changer ou interroger les modes d'un utilisateur ou d'un canal. Pour les canaux, cela peut inclure des modes tels que `+o` (opérateur), `+i` (invitation uniquement), et `+t` (sujet protégé). Pour les utilisateurs, cela peut inclure des modes tels que `+i` (invisible).

Q23 : Comment le serveur gère-t-il les limites d'utilisateurs dans un canal ?

R : Le serveur gère les limites d'utilisateurs dans un canal en utilisant le mode `+l`. Lorsqu'un utilisateur avec les permissions appropriées définit ce mode avec une limite, le nombre d'utilisateurs dans le canal est limité. La méthode `setUserLimit()` de la classe `Channel` est utilisée pour définir cette limite.

Q24 : Comment les commandes sont-elles traitées et dispatchées aux fonctions appropriées ?

R : Les commandes sont traitées et dispatchées aux fonctions appropriées par la fonction `handleCommand()` définie dans `irc_commands.cpp`. Cette fonction analyse la commande reçue et appelle la fonction de traitement correspondante, telle que `handlePing()`, `handleJoinCommand()`, ou `handlePrivmsgCommand()`.

Q25 : Que fait la méthode `broadcast()` de la classe `Channel` ?

R : La méthode `broadcast()` de la classe `Channel` envoie un message à tous les utilisateurs du canal, sauf à l'utilisateur spécifié comme expéditeur (pour éviter l'écho). Elle utilise la fonction `send()` pour envoyer le message à chaque socket client.

Q26 : Comment le serveur gère-t-il les erreurs d'authentification ?

R : Le serveur gère les erreurs d'authentification en vérifiant le mot de passe fourni par le client contre le mot de passe du serveur. Si le mot de passe est incorrect, une erreur est renvoyée au client, et la connexion peut être fermée. La méthode `setSentAuthError()` de la classe `Client` est utilisée pour marquer qu'une erreur d'authentification a été envoyée.

Q27 : Que se passe-t-il lorsqu'un utilisateur envoie une commande inconnue ?

R : Lorsqu'un utilisateur envoie une commande inconnue, le serveur répond avec un message d'erreur indiquant que la commande n'est pas reconnue. La fonction `handleCommand()` détecte les commandes inconnues et utilise la fonction `sendToClient()` pour envoyer le message d'erreur.

Q28 : Comment le serveur gère-t-il les opérateurs de canal et leurs permissions ?

R : Le serveur gère les opérateurs de canal et leurs permissions en utilisant des méthodes telles que `addOperator()`, `removeOperator()`, et `isOperator()` de la classe `Channel`. Ces méthodes permettent de gérer la liste des opérateurs et de vérifier si un utilisateur a les permissions nécessaires pour exécuter des commandes d'opérateur.

Q29 : Qu'est-ce que la méthode `displayChannels()` de la classe `Channel` fait ?

R : La méthode `displayChannels()` de la classe `Channel` affiche une liste de tous les canaux actifs et leurs membres. Cette méthode est utilisée à des fins de debug pour vérifier l'état actuel des canaux sur le serveur.

Q30 : Comment les messages d'erreur sont-ils envoyés aux clients ?

R : Les messages d'erreur sont envoyés aux clients en utilisant la fonction `sendToClient()`. Cette fonction envoie un message spécifié au socket client, en utilisant la fonction `send()`. Les erreurs peuvent inclure des messages tels que des erreurs d'authentification, des commandes inconnues, ou des permissions insuffisantes.

Q31 : Quelle est l'importance de la méthode `updateLastPong()` dans la classe `Client` ?

R : La méthode `updateLastPong()` met à jour l'horodatage du dernier message `PONG` reçu par le client. Cela est important pour détecter les clients inactifs ou déconnectés en surveillant le délai écoulé depuis le dernier `PONG`.

Q32 : Comment le serveur gère-t-il les commandes `WHOIS` et quelles informations sont fournies ?

R : Le serveur gère la commande `WHOIS` en répondant avec des informations sur l'utilisateur spécifié, telles que le pseudonyme, l'adresse utilisateur et le statut. La fonction `handleWhoisCommand()` envoie les détails pertinents au client qui a émis la commande.

Q33 : Comment les informations des utilisateurs sont-elles persistées et utilisées lors des commandes `NAMES` et `LIST` ?

R : Les informations des utilisateurs et des canaux sont stockées dans des structures de données en mémoire (`std::map`). Les commandes `NAMES` et `LIST` accèdent à ces structures pour récupérer et afficher les informations pertinentes sur les utilisateurs et les canaux.

Q34 : Quelle est la différence entre les commandes `NAMES` et `WHOIS` ?

R : La commande `NAMES` affiche la liste des utilisateurs présents dans un canal, tandis que la commande `WHOIS` fournit des informations détaillées sur un utilisateur spécifique, y compris son pseudonyme, son nom réel et son statut.

Q35 : Comment le serveur traite-t-il les déconnexions volontaires et involontaires des utilisateurs ?

R : Le serveur traite les déconnexions volontaires lorsque les utilisateurs envoient la commande `QUIT`. Les déconnexions involontaires sont détectées par la fermeture du socket client. Dans les deux cas, le serveur supprime l'utilisateur des canaux et ferme la connexion.

Q36 : Comment les modes de canal sont-ils modifiés et quelles sont leurs implications ?

R : Les modes de canal sont modifiés à l'aide de la commande `MODE`. Les modes peuvent inclure `+o` pour opérateur, `+i` pour invitation uniquement, et `+t` pour sujet protégé. Chaque mode a des implications sur les permissions et les actions autorisées dans le canal.

Q37 : Pourquoi est-il important de définir les sockets en mode non-bloquant, et comment cela est-il réalisé dans le code ?

R : Il est important de définir les sockets en mode non-bloquant pour que le serveur puisse traiter plusieurs connexions simultanément sans se bloquer sur une opération d'entrée/sortie. Cela est réalisé en utilisant `fcntl(fd, F_SETFL, O_NONBLOCK);`.

Q38 : Comment le serveur vérifie-t-il les caractères invalides dans les pseudonymes ?

R : Le serveur vérifie les caractères invalides dans les pseudonymes en utilisant la fonction `containsInvalidCharacters()` dans `utils.cpp`, qui vérifie la présence de caractères non autorisés dans le pseudonyme.

Q39 : Quels sont les rôles des structures `pollfd` et comment sont-elles utilisées dans le serveur ?

R : Les structures `pollfd` sont utilisées pour surveiller plusieurs descripteurs de fichiers afin de détecter les événements d'entrée/sortie. Dans le serveur, elles sont utilisées avec `poll()` pour surveiller les sockets clients et le socket du serveur.

Q40 : Comment le serveur gère-t-il les canaux protégés par mot de passe ?

R : Le serveur gère les canaux protégés par mot de passe en utilisant le mode `+k`. Lorsqu'un canal est protégé par mot de passe, les utilisateurs doivent fournir le mot de passe correct pour rejoindre le canal. La méthode `checkKey()` de la classe `Channel` est utilisée pour vérifier le mot de passe fourni.

Q41 : Quelle est l'importance de la commande `LIST` et comment est-elle traitée ?

R : La commande `LIST` est utilisée pour obtenir une liste de tous les canaux disponibles sur le serveur ainsi que le nombre d'utilisateurs dans chaque canal. Elle est traitée par la fonction `handleListCommand()` qui compile les informations et les envoie au client demandeur.

Q42 : Comment le serveur gère-t-il la protection des sujets de canal avec le mode `+t` ?

R : Le serveur gère la protection des sujets de canal en utilisant le mode `+t`, ce qui signifie que seuls les opérateurs du canal peuvent changer le sujet. Si un utilisateur non-opérateur essaie de changer le sujet, une erreur est renvoyée.

Q43 : Comment les utilisateurs peuvent-ils quitter un canal et quelles sont les implications ?

R : Les utilisateurs peuvent quitter un canal en envoyant la commande `PART <channel>`. Cela les retire du canal et met à jour la liste des membres du canal. Si l'utilisateur était le dernier membre du canal, le canal peut être supprimé.

Q44 : Comment le serveur traite-t-il les commandes `INVITE` pour inviter des utilisateurs dans un canal ?

R : Le serveur traite les commandes `INVITE` en ajoutant l'utilisateur invité à la liste des invités du canal et en envoyant une notification d'invitation à l'utilisateur. Cela permet à l'utilisateur invité de rejoindre le canal même s'il est en mode invitation uniquement (`+i`).

Q45 : Que se passe-t-il lorsque le serveur reçoit une commande mal formée ou invalide ?

R : Lorsqu'une commande mal formée ou invalide est reçue, le serveur envoie un message d'erreur au client indiquant que la commande n'est pas reconnue ou mal formée. Cela empêche le serveur de traiter des commandes incorrectes et de maintenir l'intégrité de la communication.