📚 Documentation complète du projet ft_irc

📋 Table des matières

---

1. Vue d'ensemble du projet {#vue-densemble}

🎯 Objectif

ft_irc est un serveur IRC (Internet Relay Chat) écrit en C++98, implémentant une partie du protocole IRC défini dans la RFC 2812. Ce projet fait partie du cursus de l'école 42.

🔑 Fonctionnalités principales

- **Serveur TCP/IP** non-bloquant utilisant `select()`
- **Authentification** des clients avec mot de passe
- **Gestion des nicknames** avec validation selon RFC 2812
- **Commandes de base** : PASS, NICK, USER, PING, QUIT
- **Multi-clients** : gestion simultanée de plusieurs connexions
- **Timeout** : déconnexion automatique des clients inactifs

📊 État actuel

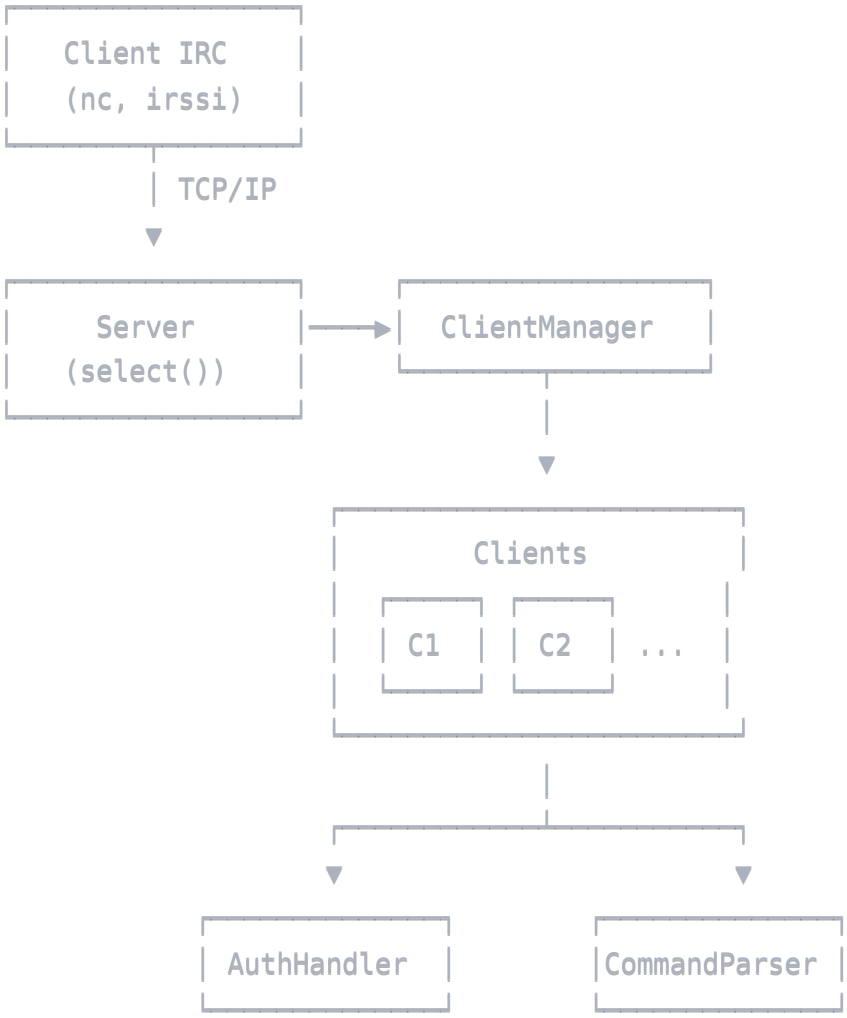
- ✅ Partie 1 : Serveur TCP/IP de base
- ✅ Partie 2 : Authentification et gestion des clients

- ❌ Partie 3 : Canaux et messagerie (non implémenté)

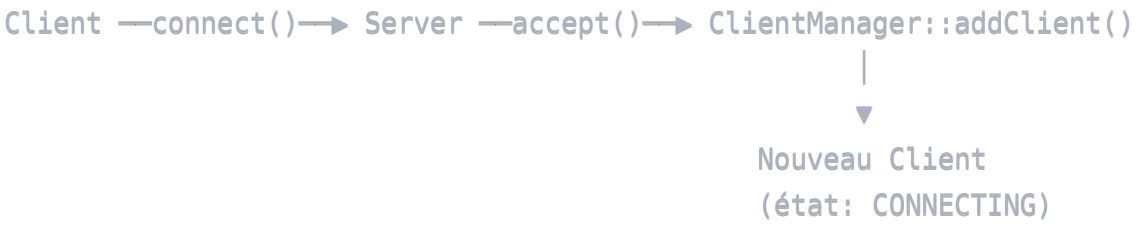
---

2. Architecture du système {#architecture}

🏗️ Architecture générale



🔄 Flux de connexion



---

3. Guide d'installation {#installation}

📋 Prérequis

- **Compilateur** : g++ ou clang++ avec support C++98
- **Système** : Linux ou macOS

- **Make** : GNU Make 3.81+

🛠️ Compilation

bash

```bash
# Cloner le projet
git clone [URL_DU_PROJET]
cd ft_irc

# Compiler
make                # Compilation normale
make debug          # Avec symboles de debug
make re             # Recompiler entièrement

# Nettoyer
make clean          # Supprimer les .o
make fclean         # Supprimer tout
```

🚀 Lancement

bash

```bash
# Format : ./ft_irc <port> <password>
./ft_irc 6667 mypassword

# Exemple avec port personnalisé
./ft_irc 8080 secret123
```

---

4. Structure des fichiers {#structure-fichiers}

```
ft_irc/
│
├── Makefile                    # Système de compilation
├── test_part2.sh               # Script de tests automatisés
│
├── Client.hpp                  # Définition de la classe Client
├── Client.cpp                  # Implémentation Client
│
├── ClientManager.hpp           # Gestionnaire de clients
├── ClientManager.cpp           # Implémentation
│
├── AuthHandler.hpp             # Gestion de l'authentification
├── AuthHandler.cpp             # Implémentation
│
├── CommandParser.hpp           # Parseur de commandes IRC
├── CommandParser.cpp           # Implémentation
│
└── server_integration_example.cpp  # Serveur principal avec main()
```

📝 Description des fichiers

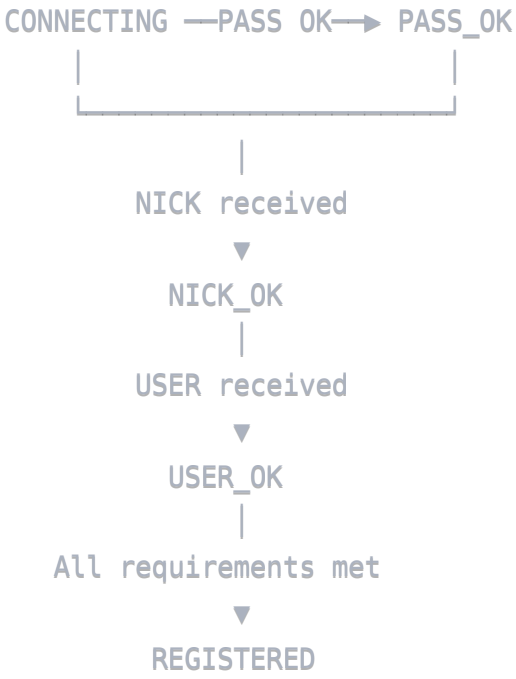
Fichier	Rôle	Dépendances
Client	Représente un client connecté	Aucune
ClientManager	Gère tous les clients	Client, AuthHandler, CommandParser
AuthHandler	Authentification IRC	Client
CommandParser	Parse et exécute les commandes	Client, AuthHandler
Server	Boucle principale du serveur	ClientManager

---

5. Diagrammes de classes {#diagrammes}

📐 Diagramme UML simplifié

🔄 États d'un client



---

6. Flux de données {#flux-donnees}

📨 Réception de données

```
1. Socket recv() ──► Server::handleClientData()
                                  │
2. Données brutes ──────────────► Client::appendToBuffer()
                                  │
3. Buffer client ──────────────► Client::extractMessage()
                                  │
4. Message complet ────────────► CommandParser::processMessage()
                                  │
5. Commande parsée ────────────► Handler approprié
```

📤 Envoi de données

```
1. Handler génère réponse ──► Client::sendMessage()
                                        │
2. Ajout de \r\n ──────────────────────► send() sur socket
```

---

7. Protocole IRC implémenté {#protocole-irc}

📜 Commandes supportées

Commande	Format	Description	État requis
PASS	`PASS <password>`	Authentification serveur	CONNECTING
NICK	`NICK <nickname>`	Définir le pseudo	PASS_OK
USER	`USER <user> <mode> <unused> :<realname>`	Infos utilisateur	NICK_OK
PING	`PING :<token>`	Test de connexion	REGISTERED
QUIT	`QUIT :[message]`	Déconnexion	Tous

📊 Codes de réponse

Code	Nom	Description
001	RPL_WELCOME	Message de bienvenue
002	RPL_YOURHOST	Info sur le serveur
003	RPL_CREATED	Date de création
004	RPL_MYINFO	Info serveur
421	ERR_UNKNOWNCOMMAND	Commande inconnue
431	ERR_NONICKNAMEGIVEN	Pas de nickname
432	ERR_ERRONEUSNICKNAME	Nickname invalide
433	ERR_NICKNAMEINUSE	Nickname déjà utilisé
451	ERR_NOTREGISTERED	Non enregistré
461	ERR_NEEDMOREPARAMS	Paramètres manquants
462	ERR_ALREADYREGISTERED	Déjà enregistré
464	ERR_PASSWDMISMATCH	Mot de passe incorrect

---

8. Guide détaillé des classes {#classes-detail}

🔵 Classe Client

Rôle : Représente un client connecté au serveur

Attributs principaux :

- `_fd` : File descriptor du socket
- `_nickname` : Pseudo IRC
- `_username` : Nom d'utilisateur
- `_realname` : Nom réel
- `_state` : État actuel (enum ClientState)
- `_buffer` : Buffer de réception
- `_passwordOk` : Mot de passe validé

- `_lastActivity` : Timestamp dernière activité

Méthodes clés :

```cpp
// Gestion du buffer
void appendToBuffer(const std::string& data);  // Ajoute au buffer
std::string extractMessage();                  // Extrait un message complet

// Communication
void sendMessage(const std::string& msg);      // Envoie au client
std::string getPrefix();                       // Format: :nick!user@host

// État
bool isRegistered();                           // Client complètement connecté
bool isTimedOut(int timeout);                  // Vérifie timeout
```

🟢 Classe ClientManager

Rôle : Gestionnaire central de tous les clients

Responsabilités :

- Créer/supprimer des clients
- Router les données vers le bon handler
- Gérer les timeouts
- Maintenir la liste des clients

Méthodes importantes :

```cpp
void addClient(int fd);                        // Nouveau client
void removeClient(int fd);                     // Déconnexion
void handleClientData(int fd, const string& data); // Données reçues
void checkTimeouts();                          // Vérification périodique
```

🔴 Classe AuthHandler

Rôle : Gère l'authentification IRC

Fonctionnalités :

- Valide les mots de passe
- Vérifie les nicknames (format, unicité)

- Gère l'enregistrement des clients
- Envoie les messages de bienvenue

Validation nickname :

- Max 9 caractères
- Commence par lettre ou `[]\{}|_`
- Contient : alphanumériques + `[]\{}|_-`

🟡 Classe CommandParser

Rôle : Parse et exécute les commandes IRC

Structure d'un message IRC :

```
[:prefix] COMMAND [params] [:trailing param]

Exemples :
NICK alice
:alice!user@host PRIVMSG #chan :Hello world
```

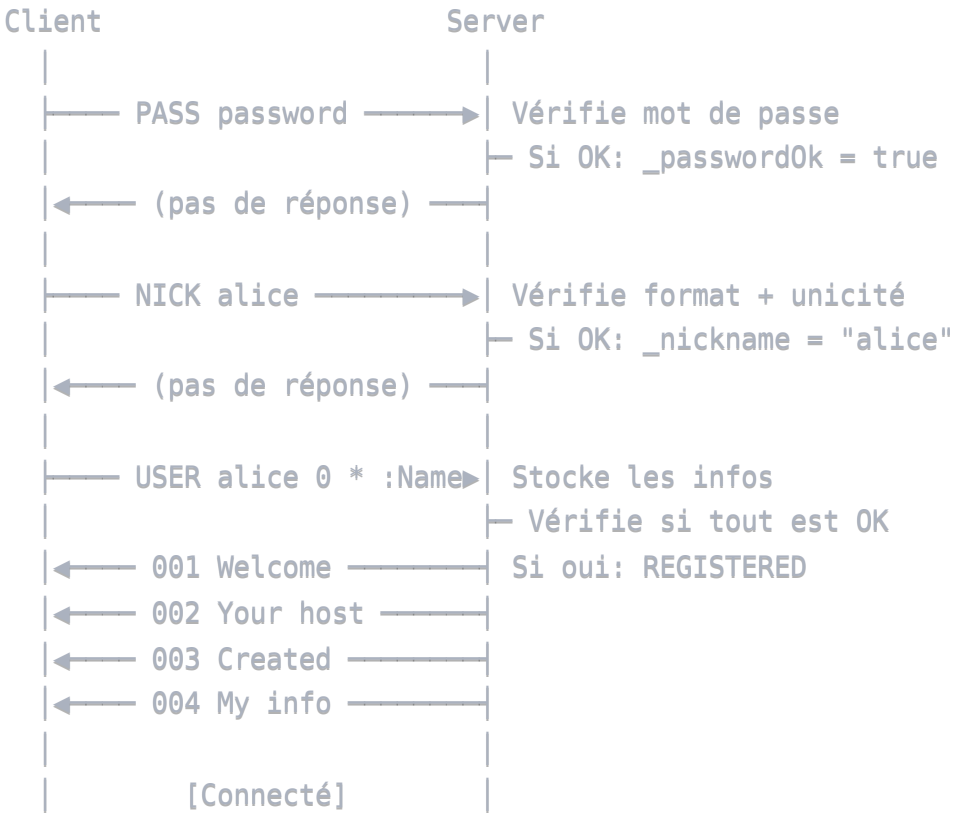
Processus :

1. Extraction du préfixe (optionnel)
2. Extraction de la commande
3. Parsing des paramètres
4. Routage vers le bon handler

---

9. Processus d'authentification {#authentification}

🔐 Séquence complète



⚠️ Règles importantes

1. **Ordre** : PASS doit être envoyé en premier
2. **Unicité** : Chaque nickname doit être unique
3. **Validation** : Les nicknames invalides sont rejetés
4. **Complet** : Les 3 commandes sont nécessaires

---

10. Gestion des erreurs {#gestion-erreurs}

🛡️ Types d'erreurs gérées

Type	Exemple	Traitement
Socket	`recv()` retourne -1	Déconnexion client
Protocole	Commande inconnue	Erreur 421
Validation	Nickname invalide	Erreur 432
État	Commande sans auth	Erreur 451
Timeout	Inactivité 300s	Déconnexion auto

📝 Logging

cpp

```cpp
// Connexion
std::cout << "New client connected (fd: " << fd << ")" << std::endl;

// Authentification
std::cout << "Client " << nick << " registered" << std::endl;

// Déconnexion
std::cout << "Client " << nick << " disconnected" << std::endl;
```

---

11. Tests et validation {#tests}

🧪 Script de test automatisé

bash

```bash
# Lancer tous les tests
./test_part2.sh

# Tests individuels
./test_part2.sh 6667 mypassword
```

✅ Checklist de validation

- [ ] Compilation sans warnings
- [ ] Pas de fuites mémoire (valgrind)
- [ ] Authentification fonctionnelle
- [ ] Gestion multi-clients
- [ ] Nicknames validés correctement
- [ ] Timeouts fonctionnels
- [ ] Déconnexion propre
- [ ] Résistance aux entrées invalides

🔍 Tests manuels recommandés

bash

```
# Test basique
nc localhost 6667
PASS password
NICK test
USER test 0 * :Test User

# Test avec telnet
telnet localhost 6667

# Test avec client IRC
irssi -c localhost -p 6667 -w password
```

---

12. Exemples d'utilisation {#exemples}

💻 Exemple 1 : Connexion simple

bash

```
$ nc localhost 6667
NOTICE AUTH :*** Looking up your hostname...
NOTICE AUTH :*** Found your hostname
PASS secret123
NICK alice
USER alice 0 * :Alice Smith
:ft_irc.42.fr 001 alice :Welcome to the Internet Relay Network alice!alice@localhost
:ft_irc.42.fr 002 alice :Your host is ft_irc.42.fr, running version ft_irc-1.0
:ft_irc.42.fr 003 alice :This server was created by 42 students
:ft_irc.42.fr 004 alice :ft_irc.42.fr ft_irc-1.0 o o
PING :test
PONG :test
QUIT :Bye
ERROR :Bye
```

🚫 Exemple 2 : Erreurs communes

bash

```
# Mauvais mot de passe
PASS wrong
NICK test
:ft_irc.42.fr 464 * :Password incorrect

# Nickname invalide
PASS secret123
NICK 123test
:ft_irc.42.fr 432 * 123test :Erroneous nickname

# Nickname déjà pris
PASS secret123
NICK alice
:ft_irc.42.fr 433 * alice :Nickname is already in use
```

---

13. Dépannage {#depannage}

❌ Problèmes courants

Problème	Cause possible	Solution
"Address already in use"	Port déjà utilisé	Changer de port ou `killall ft_irc`
Connexion refusée	Serveur pas lancé	Vérifier que le serveur tourne
Timeout immédiat	Buffer mal géré	Vérifier `extractMessage()`
Pas de réponse	Commandes mal parsées	Debug dans `CommandParser`
Segfault	Pointeur null	Vérifier avec valgrind

🔧 Debug avec GDB

bash

```
# Compiler en mode debug
make debug

# Lancer avec GDB
gdb ./ft_irc
(gdb) run 6667 password
(gdb) break ClientManager::handleClientData
(gdb) continue
```

📊 Monitoring

bash

```bash
# Voir les connexions actives
netstat -an | grep 6667

# Suivre l'activité réseau
tcpdump -i lo -A port 6667

# Vérifier l'utilisation mémoire
valgrind --leak-check=full ./ft_irc 6667 pass
```

🎓 Conclusion

Ce serveur IRC implémente les bases du protocole IRC avec une architecture modulaire et extensible. La partie authentification est complète et robuste, prête pour l'ajout des fonctionnalités de messagerie et de canaux.

📚 Ressources supplémentaires

- RFC 2812 - IRC Protocol
- Modern IRC Documentation
- IRC Numerics Reference

👥 Contributeurs

Projet réalisé dans le cadre du cursus 42.