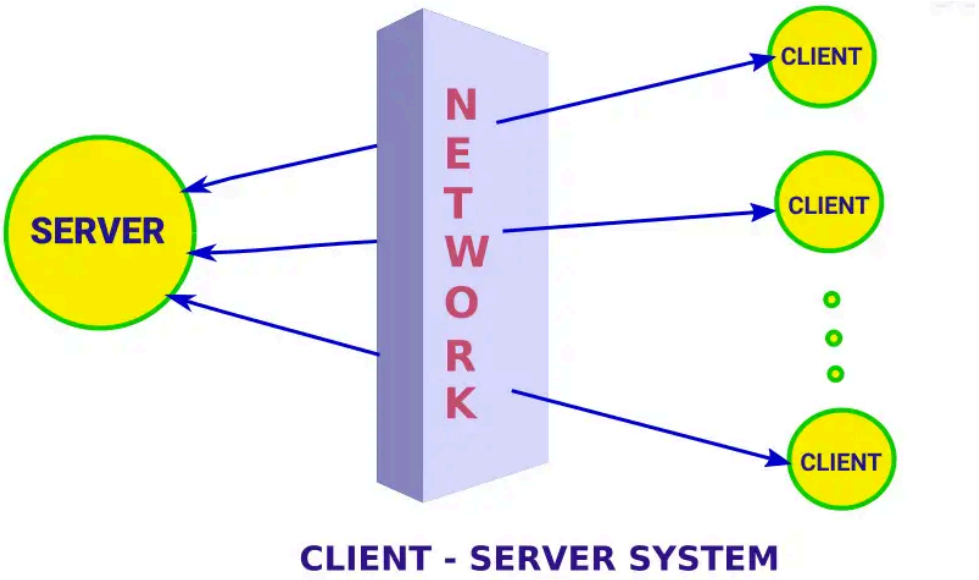
Projet - OS USER

Sherlock 13



EI4

2024/2025

Inessa Kechek (Groupe TPA)

Table des matières

1. Introduction

Ce projet est une adaptation réseau du jeu de société Sherlock 13, un jeu de déduction où chaque joueur doit identifier un suspect à l'aide d'indices et de cartes. Il a été implémenté en C, sous forme d'un programme client-serveur permettant à 4 joueurs de se connecter à un serveur central et d'interagir à travers une interface graphique développée avec SDL2. Chaque joueur dispose de cartes représentant différents personnages, chacun associé à des indices (objets). Le but est de déterminer quelle carte personnage a été désignée comme "coupable" en posant des questions aux autres joueurs sur les indices qu'ils possèdent.

L'objectif était d'exploiter des concepts fondamentaux de la programmation réseau et concurrente, notamment les sockets TCP, les threads et la synchronisation à l'aide de mutex.

2. Architecture generale

L'architecture repose sur un modèle centralisé : un serveur coordonne les connexions des clients, distribue les cartes et gère le déroulement de la partie. Les clients se connectent au serveur, attendent les messages pour effectuer leurs actions et envoient les informations de ce qu'ils veulent faire en interagissant avec l'interface SDL.

a. Sockets TCP

On a utilisé des sockets TCP pour établir la communication entre le serveur et les clients. Les sockets TCP ont été choisis pour leur fiabilité dans la transmission des données, garantissant que les messages échangés entre les joueurs et le serveur sont correctement délivrés dans l'ordre d'envoi, ce qui est crucial pour la synchronisation d'un jeu multijoueur.

Dans ce projet :

- Le serveur crée un socket d'écoute (**sockfd**) qui attend les connexions entrantes sur un port spécifié en argument (./server 8080 par exemple).

- Chaque client possède deux types de sockets :
  - Un socket client pour envoyer des messages au serveur (**sendMessageToServer**)
  - Un socket serveur qui s'exécute dans un thread séparé pour recevoir les messages du serveur (**fn_serveur_tcp**)

Les fonctions principales utilisées pour gérer les sockets sont :

- **socket()** pour créer un nouveau socket
- **bind()** pour attacher le socket à une adresse et un port
- **listen()** pour mettre le socket en mode écoute
- **accept()** pour accepter les connexions entrantes
- **connect()** pour établir une connexion avec un serveur distant
- **read()** et **write()** pour l'échange de données

b. Threads

Le multithreading est un aspect essentiel de ce jeu pour permettre une communication asynchrone. Dans le client, un thread dédié est créé pour écouter en permanence les messages venant du serveur, pendant que le thread principal gère l'interface utilisateur et les interactions de l'utilisateur.

Le thread serveur TCP (**thread_serveur_tcp_id**) est créé au démarrage du client avec la fonction **pthread_create()**. Ce thread exécute la fonction **fn_serveur_tcp()** qui :

- Crée un socket serveur
- Lie ce socket à l'adresse IP et au port du client
- Se met en écoute des connexions entrantes
- Accepte les connexions et lit les messages reçus
- Signale au thread principal qu'un message a été reçu via la variable synchro

Cette approche permet au client de rester réactif aux entrées utilisateur tout en étant capable de recevoir et traiter les messages du serveur à tout moment.

c. Mutex

La synchronisation entre threads est principalement gérée par la variable **synchro** qui sert de drapeau pour indiquer qu'un nouveau message est disponible.
Cette variable est mise à 1 par le thread serveur lorsqu'un message est reçu, puis remise à 0 par le thread principal après traitement du message.

3. Fonctionnement global du programme

a. Serveur

Le serveur (server.c) joue un rôle central dans la coordination de la partie. Il est responsable de:

i.Initialisation du jeu :

- Création du plateau de jeu avec **createTable() :**

```
0 Sebastian Moran
1 irene Adler
2 inspector Lestrade
3 inspector Gregson
4 inspector Baynes
5 inspector Bradstreet
6 inspector Hopkins
7 Sherlock Holmes
8 John Watson
9 Mycroft Holmes
10 Mrs. Hudson
11 Mary Morstan
12 James Moriarty
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

- Mélange des cartes avec **melangerDeck() :**

```
5 inspector Bradstreet
6 inspector Hopkins
2 inspector Lestrade
12 James Moriarty
9 Mycroft Holmes
4 inspector Baynes
8 John Watson
11 Mary Morstan
3 inspector Gregson
10 Mrs. Hudson
0 Sebastian Moran
1 irene Adler
7 Sherlock Holmes
01 00 01 03 01 00 02 00
01 03 00 01 01 00 00 01
01 00 02 01 02 01 01 00
01 01 01 00 00 02 00 02
```

- Attente de la connexion des quatre joueurs

ii.Gestion des connexions clients :

- Enregistrement des informations des clients (IP, port, nom) dans la **structure tcpClients**
- Attribution d'un **identifiant unique** à chaque joueur :

```
Received packet from 127.0.0.1:55636
Data: [C 127.0.0.1 9090 inessa
]

COM=C ipAddress=127.0.0.1 port=9090 name=inessa
0: 127.0.0.1 09090 inessa
id=0
```

- Diffusion de la **liste des joueurs** à tous les clients :

```
┌──(iness㉿LAPTOP-K51SLL86)-[/mnt/c/Users/iness/Desktop/OS_USER/sh13_etu]
└─$ ./bin/sh13 127.0.0.1 8080 127.0.0.1 9090 inessa
Sans=0x5620e7c64f00
Creation du thread serveur tcp !
consomme |I 0
|
consomme |L inessa - - -
|
```

iii.Distribution des cartes :

- Attribution des cartes aux joueurs (3 cartes par joueur)
- Envoi à chaque joueur de ses cartes et de ses indices

```
sprintf(reply,"D %d %d %d", deck[0], deck[1], deck[2]);
sendMessageToClient(tcpClients[0].ipAddress,tcpClients[0].port, reply);
for (int j = 0; j < 8; j++) {
    sprintf(reply, "V %d %d", j, tableCartes[0][j]);
    sendMessageToClient(tcpClients[0].ipAddress, tcpClients[0].port, reply);
}
```

iv.Gestion du tour par tour :

- Définition du joueur courant
- Traitement des différentes actions des joueurs (questions, suppositions) : ‘G’, ‘O’ et ‘S’

```
Received packet from 127.0.0.1:56568
Data: [O 0 3
]

Player 0 asks all of the players for clue index 3
```

```
Received packet from 127.0.0.1:43520
Data: [S 2 1 6
]

Player 2 asks for clue index 6 from Player 1
```

- Passage au joueur suivant après chaque action

v.Vérification des conditions de fin de partie :

- Identification d'une supposition correcte (victoire)
- Détection des joueurs éliminés

```
Received packet from 127.0.0.1:44470
Data: [G 1 6
]

Player 1 guessed the culprit: 6 (inspector Hopkins)
Player 1 failed the guess.
```

```
Received packet from 127.0.0.1:35338
Data: [G 2 0
]

Player 2 guessed the culprit: 0 (Sebastian Moran)
Player 2 wins the game!
```

Le serveur utilise plusieurs types de messages pour communiquer avec les clients, identifiés par une lettre en début de message ('I', 'L', 'D', etc.) et suivis de paramètres spécifiques.

b. Client

Le client (sh13.c) est responsable de :

i.Interface utilisateur :

- Affichage du plateau de jeu avec SDL2

- Gestion des entrées utilisateur (souris pour la sélection des joueurs, objets, coupables)

	5	5	5	5	4	3	3	3
inessa	1	0	3	1	0	0	1	1
marie								
alex								
bob								

```
case  SDL_MOUSEBUTTONDOWN:
    SDL_GetMouseState( &mx, &my );
    //printf("mx=%d my=%d\n",mx,my);
    if ((mx<200) && (my<50) && (connectEnabled==1))···
    else if ((mx>=0) && (mx<200) && (my>=90) && (my<330))···
    else if ((mx>=200) && (mx<680) && (my>=0) && (my<90))···
    else if ((mx>=100) && (mx<250) && (my>=350) && (my<740))···
    else if ((mx>=250) && (mx<300) && (my>=350) && (my<740))···
    else if ((mx>=500) && (mx<700) && (my>=350) && (my<450) && (goEnabled==1))···
    else
```

- Visualisation des cartes du joueur et des informations collectées

ii.Communication avec le serveur :

- Envoi des actions du joueur via sendMessageToServer()
- Réception et traitement des messages du serveur dans la fonction exécutée par le thread serveur TCP

```
consomme |V 0 1
|
consomme |V 1 0
|
consomme |V 2 1
|
consomme |V 3 3
|
consomme |V 4 1
|
consomme |V 5 0
|
consomme |V 6 2
|
consomme |V 7 0
|
```

```
consomme |M 0
|
```

```
consomme |D 5 6 2
|
```

```
go! joueur=-1 objet=4 guilt=-1
consomme |R 0 0
|
consomme |R 0 2
|
consomme |R 1 3
|
```

iii.Logique de jeu locale :

- Suivi de l'état du jeu (tour actuel, cartes, informations collectées)
- Mise à jour de l'interface en fonction des messages reçus
- Gestion des règles locales (sélection des actions possibles)

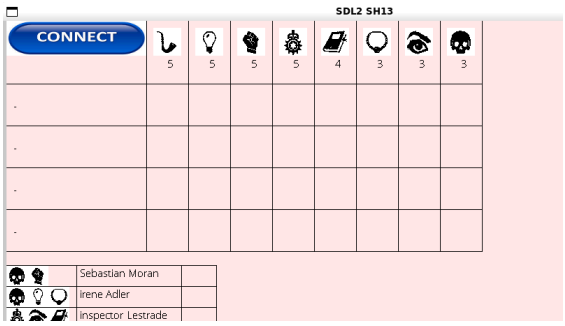
Le client implémente plusieurs traitements de messages en fonction de leur type :

I <id>	réception de l'identifiant du joueur
D <cart1> <carte2> <carte3>	réception des cartes distribuées
V <colonne> <valeur>	mise à jour de la table des indices
R 1 <id>	Si le joueur pose une question à tous les autres joueurs, il reçoit cette réponse si le joueur avec id possède l'objet
R 0 <id>	Si le joueur pose une question à tous les autres joueurs, il reçoit cette réponse si le joueur avec id ne possède pas l'objet
R <x> <id>	Si le joueur pose une question à un joueur spécifique, il reçoit cette réponse indiquant que le joueur avec id possède x nombre de l'objet
E	fin de partie (tous éliminés)
W <id> <nomCoupable>	Le joueur avec l'id à deviner correctement le coupable et a gagné
F <id>	Le joueur avec l'id a essayé de deviner le coupable mais a échoué (celui ci reçoit aussi le nom du coupable mais ceci n'est pas diffusé aux autres)

c. Déroulement d'une partie

i.Phase d'initialisation :

- Lancement du serveur avec un port spécifié
- Connexion des quatre joueurs au serveur via le bouton "Connect"



- Le serveur attribue des identifiants aux joueurs et diffuse la liste des participants
- Distribution des cartes et des indices

ii.Phase de jeu :

- Le serveur désigne le premier joueur (joueur 0)
- À son tour, un joueur peut :

1. Interroger tous les joueurs sur un objet spécifique (bouton "Go" après sélection d'un objet)

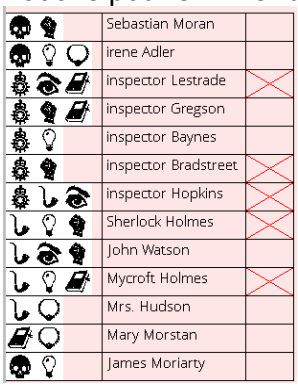
2. Interroger un joueur spécifique sur un objet (bouton "Go" après sélection d'un joueur et d'un objet)

3. Faire une supposition sur l'identité du coupable (bouton "Go" après sélection d'un personnage)

iii.Réponses aux questions :

Quand un joueur interroge sur un objet, le serveur consulte la table des indices (**tableCartes**) et envoie les réponses au joueur qui a posé la question.

Le joueur peut utiliser ces informations pour éliminer des suspects (en cliquant dans la colonne à droite des noms).



iv.Fin de partie :

- Si un joueur identifie correctement le coupable, il gagne et la partie se termine (**exit(0)**)
- Si un joueur fait une supposition incorrecte, il est éliminé (**elimine[id]=1**)
- Si tous les joueurs sont éliminés, la partie se termine sans vainqueur (**exit(0)**)

4. Conclusion

Ce projet illustre l'application de plusieurs concepts fondamentaux de programmation réseau :

- Communication réseau via les **sockets TCP** pour permettre l'interaction entre plusieurs clients et un serveur central.
- Programmation concurrente avec l'**utilisation de threads** pour gérer simultanément l'interface utilisateur et la communication réseau.
- **Architecture client-serveur** où le serveur joue le rôle de coordinateur central tandis que les clients sont responsables de l'interface utilisateur et des interactions locales.

Les aspects qui pourraient être améliorés incluent une gestion plus robuste des erreurs et l'ajout de fonctionnalités comme la reconnexion des joueurs en cas de déconnexion ou une interface utilisateur plus élaborée.