Projet : Sherlock 13 Multijoueur
Nom : Yedam KIM
Date : 20/04/2025
Module : OS_User

Rapport de Projet: Sherlock 13

1. Introduction

Le projet proposé dans le cadre du module OS USER consistait à développer une application réseau multijoueur permettant de jouer au jeu Sherlock 13, en respectant une architecture client/serveur, et en intégrant des aspects vus en TP tels que :

- les sockets TCP
- la gestion des processus/threads
- la synchronisation avec des mutex
- l'utilisation de bibliothèques multimédia (SDL2, TTF, SDL_Image)

Ce projet visait à mettre en pratique nos acquis sur la communication entre processus distants, l'implémentation de protocoles, et l'usage d'un moteur graphique bas niveau. Il s'agissait également d'intégrer des notions fondamentales sur les appels système, la gestion de processus, les échanges inter-processus, et de simuler la complexité d'un système distribué interactif.

2. Description du Jeu Sherlock 13

Sherlock 13 est un jeu de déduction dans lequel les joueurs doivent identifier un criminel caché parmi treize personnages, chacun étant associé à un ensemble unique de symboles (loupe, pipe, collier, etc.). Au début de la partie, une carte est placée face cachée au centre : c'est le criminel. Les douze autres cartes sont réparties entre les joueurs : quatre cartes chacun à trois joueurs, trois cartes chacun à quatre joueurs.

Chaque joueur dispose d'une feuille d'enquête pour croiser les informations : en haut, il peut noter les noms des autres joueurs et le nombre total de symboles qu'ils possèdent ; en bas, un tableau croisé liste les personnages et les symboles correspondants. Les joueurs utilisent cette feuille pour éliminer des suspects au fur et à mesure des tours.

À son tour, un joueur peut effectuer une seule action parmi trois :

1. **Enquête globale** : il choisit un symbole et demande aux autres joueurs s'ils le possèdent. Ceux qui l'ont lèvent la main.

2. **Enquête ciblée** : il choisit un joueur et un symbole ; ce joueur annonce combien de fois ce symbole est présent dans sa main.

3. **Accusation** : il annonce le nom du coupable. Si c'est correct, il gagne. Sinon, il est éliminé et ne joue plus.

Le jeu se termine lorsqu'un joueur réussit une accusation ou que tous les autres ont échoué, auquel cas le dernier joueur restant remporte la partie.

Une variante « Expert » introduit une contrainte supplémentaire : les cartes doivent être classées en main, et les symboles de la dernière carte ne comptent pas dans les réponses. Cela ajoute une dimension stratégique au placement et à la gestion des informations.

3. Architecture du Projet

Le projet repose sur une architecture client/serveur classique. Le serveur centralise la logique du jeu, gère les connexions TCP des joueurs, distribue les cartes et coordonne les tours à l'aide d'une machine à états. Les messages sont codés en ASCII, lisibles et faciles à parser. Le serveur tient à jour les états (cartes, joueur actif, éliminations) et répond aux requêtes (questions, accusations).

Le client (sh13.c) utilise SDL2 pour afficher les cartes, les objets et recevoir les interactions utilisateur. Il envoie des messages réseau correspondant aux actions, et un thread écoute les réponses du serveur pour mettre à jour l'interface. La synchronisation entre ce thread et l'affichage principal est gérée par un mutex. Cette structure modulaire permet une bonne clarté du code et une extension facile.

4. Fonctionnement et Protocole

Le jeu fonctionne via un protocole ASCII simple : le serveur distribue les cartes une fois les joueurs connectés, et coordonne les échanges selon les actions. Chaque joueur interagit via une interface graphique en cliquant sur des objets, personnages ou boutons, générant des messages envoyés au serveur. Celui-ci traite la requête et envoie une réponse mise à jour à tous les clients.

Côté client, un thread TCP dédié écoute les messages du serveur en continu pour éviter de bloquer l'interface SDL. Les clics sont interprétés via des SDL_Rect, et les messages sont affichés dynamiquement. La synchronisation est assurée par un mutex pour garantir la cohérence des données réseau et graphiques.

Projet : Sherlock 13 Multijoueur
Nom : Yedam KIM
Date : 20/04/2025
Module : OS_User

5. Intégration des concepts du module OS USER

Le projet intègre les notions clés du module OS USER :

- **Sockets TCP** pour la communication client-serveur, avec les fonctions classiques socket, bind, listen, accept, connect, read, write.
- **Threads** pour permettre à l'interface de rester réactive pendant les communications réseau.
- **Mutex** pour éviter les conflits d'accès aux buffers partagés.
- **Protocoles ASCII** par TPs, simples à construire et à analyser.
- **SDL2 / SDL_TTF** pour l'interface graphique, gestion d'événements, rendu d'images et de texte.
- **Appels système Unix** (open, close, etc.) utilisés en bas niveau.

La structure choisie évite l'utilisation de fork ou pipe en production, mais leur usage étudié a renforcé la conception générale. Ce projet met en œuvre de manière cohérente les principaux savoir-faire du module OS USER.

6. Démarche de développement

Pour concevoir ce programme, j'ai commencé par étudier attentivement l'énoncé du projet et le protocole de communication attendu. Ensuite, j'ai organisé le projet en deux grandes parties : le serveur (logique de jeu, synchronisation des joueurs) et le client (affichage, interaction utilisateur, communication).

La complétion du code C fourni a nécessité une compréhension fine des messages échangés, de l'encodage ASCII, et des structures de données partagées côté serveur. J'ai dû concevoir la table de jeu, écrire les fonctions de distribution, puis gérer les différentes requêtes des clients en maintenant la cohérence des états (FSM).

J'ai commencé par une version minimaliste du serveur, en mode console, avant d'ajouter la logique complète du protocole et le support multi-clients. Côté client, j'ai utilisé les bases vues dans les TPs SDL pour créer une interface fluide, gérer les événements souris, afficher les cartes, objets, et zones interactives.

Projet : Sherlock 13 Multijoueur
Nom : Yedam KIM
Date : 20/04/2025
Module : OS_User

Chaque étape de développement était testée en parallèle avec des impressions printf, puis remplacée par des affichages SDL ou des messages réseau. J'ai appliqué des méthodes incrémentales, en isolant les fonctions critiques comme sendMessage, recvMessage, ou drawCard. La structure FSM m'a permis de bien organiser les phases du jeu (connexion, distribution, tour par tour, fin de partie).

7. Ce que j'ai fait personnellement

- Implémentation complète de sh13.c (client) :
    - Conception de l'ensemble des boucles d'affichage, dessin des textures, et rafraîchissement SDL
    - Réception, parsing et traitement des messages ASCII envoyés par le serveur
    - Mise en place de la gestion d'interactions utilisateur (clics objets, joueurs, bouton accusation)
    - Ajout de mécanismes de retour utilisateur : affichage SDL des messages reçus (lastMessage)
    - Gestion du thread TCP de réception (pthread_create) et synchronisation via mutex
    - Nettoyage du code, correction des incompatibilités systèmes (bcopy, gethostbyname)
    - Intégration des polices, images et gestion de leur chargement mémoire dynamique
- Implémentation complète de server.c :
    - Gestion des connexions clients (phase attente)
    - Moteur de jeu basé sur une FSM (machine à états)
    - Gestion du protocole ASCII complet avec parsing robuste
    - Logique de jeu : mélange des cartes, distribution, attribution des objets, gestion des éliminations, détection de victoire
    - Fiabilisation du serveur multi-clients sans conflits de threads
- Documentation et rendu :

Projet : Sherlock 13 Multijoueur
Nom : Yedam KIM
Date : 20/04/2025
Module : OS_User

- Rédaction du README.md clair et structuré pour permettre à quiconque de tester le jeu

- Vérification complète de la conformité du projet aux attentes (nombre de joueurs, scénarios de test, stabilité des connexions)

8. Difficultés rencontrées

- Blocage initial de l'interface dû à un recv() bloquant sans thread : nécessité d'utiliser une réception asynchrone

- Difficultés de mise à l'échelle graphique : gestion des zones interactives avec la souris, précision des SDL_Rect

- Fiabilisation du protocole ASCII côté serveur : erreurs de parsing lorsque les messages étaient mal formés

- Problèmes d'alignement entre les ID des joueurs, les index du tableau serveur, et les actions envoyées

- Compatibilité SDL entre machines (polices non chargées, textures incorrectes)

9. Conclusion

Ce projet m'a permis de consolider mes connaissances en programmation système, tout en me familiarisant avec la conception d'un protocole de communication, l'usage des threads, et la synchronisation via mutex. Il m'a aussi permis de développer une application multimédia avec SDL2, et de manipuler les sockets à bas niveau dans un cadre concret et ludique.
J'ai pu aborder une grande partie des concepts du module OS USER de manière approfondie et les mettre en œuvre dans un projet complet, fonctionnel et interactif. Grâce à la prise en charge de l'ensemble du pipeline de l'initialisation réseau jusqu'au rendu final à l'écran.