POLYTECH SORBONNE

Rapport projet OS User: Sherlock-13

Farah Ghliss

I. Contexte et Architecture du Jeu 2
Déroulement du jeu 3
II. Serveur 3
a. Rôle du Serveur 3
b. Code Complété et Explications 4
Lancement du Jeu (nbClients == 4) 4

Affichage du coupable pour le test 4
Gestion des Commandes Client 5
c. Aspects Techniques Utilisés 7
III. Client 8
a. Rôle du Client 8
b. Code Complété et Explications 8
Envoi des commandes (G, O, S) 8
Traitement des Réponses Serveur 9
c. Aspects Techniques Utilisés 11

I. Contexte et Architecture du Jeu

Le jeu Sherlock-13 est un jeu d'enquête basé sur une structure réseau, où quatre joueurs (clients) tentent de déterminer l'identité d'un coupable parmi 13 suspects, en utilisant des indices sur des objets associés aux suspects.

Le jeu est organisé autour d'un modèle client-serveur avec une architecture de réseau simple : un serveur central coordonne l'ensemble des échanges et du déroulement de la partie avec quatre clients connectés simultanément.

Pour tester le fonctionnement complet, le projet a été exécuté sur une seule machine, en simulant les quatre clients à l'aide de quatre terminaux distincts, chacun exécutant une instance du programme client sh13.

Chaque instance cliente ouvre une fenêtre graphique SDL contenant :

- Une grille d'objets (8 objets représentés graphiquement),
- Une liste des suspects,
- Un bouton "Connect" pour établir la connexion avec le serveur,
- Un bouton "Go" **activé uniquement pendant le tour du joueur correspondant**.

Déroulement du jeu

1. Une fois les quatre clients connectés via le bouton "Connect", le serveur distribue automatiquement trois cartes de suspects innocents à chaque joueur. Ces cartes sont affichées sur l'interface graphique.
2. Le premier joueur voit apparaître le bouton "Go" dans sa fenêtre. Il peut soit poser une question sur un objet ou un joueur, soit faire une accusation.
   Une fois l'action terminée, le tour passe au joueur suivant qui voit apparaître à son tour le bouton "Go".
3. Dès qu'un joueur découvre le coupable et envoie une accusation correcte, le serveur envoie un message de victoire à tous les clients.
   À ce moment, le bouton "Go" devient inactif sur toutes les interfaces : la partie est terminée.

II. Serveur

a. Rôle du Serveur

Le serveur gère l'ensemble du déroulement de la partie : il contrôle les connexions, distribue les cartes, actualise l'état du jeu, et détermine le vainqueur.

b. Code Complété et Explications

Lancement du Jeu (nbClients == 4)

```
                            if (nbClients == 4) {
    // 1) Envoi des 3 cartes à chaque joueur
    for (i = 0; i < 4; i++) {
        sprintf(reply, "D %d %d %d",
                deck[i*3], deck[i*3+1], deck[i*3+2]);
        sendMessageToClient(
            tcpClients[i].ipAddress,
            tcpClients[i].port,
            reply
        );
        // 2) Envoi des 8 valeurs de tableCartes
        for (int j = 0; j < 8; j++) {
            sprintf(reply, "V %d %d %d", i, j, tableCartes[i][j]);
            sendMessageToClient(
                tcpClients[i].ipAddress,
                tcpClients[i].port,
                reply
            );
        }
    }
    // 3) Annonce du joueur courant à tous
    sprintf(reply, "M %d", joueurCourant);
    broadcastMessage(reply);

    fsmServer = 1;
}
break;
}
```

Lorsque le nombre de joueurs connectés atteint quatre, le serveur :

- Envoie 3 cartes à chaque joueur.
- Initialise et envoie à chaque joueur les valeurs de la grille tableCartes, indiquant les indices sur les objets.
- Informe tous les joueurs du joueur courant, permettant à ce dernier d'effectuer sa première action.

Cette partie du code permet d’assurer que tous les joueurs disposent des mêmes informations initiales.

Affichage du coupable pour le test

Nous avons ajouté une ligne qui affiche le nom du coupable immédiatement après son tirage pour faciliter les tests :

```
    printf("TEST : carte coupable désignée = « %s » (indice %d)\n",
```

Cette ligne nous a permis plus tard de vérifier rapidement la logique du jeu en affichant directement le suspect coupable sur le terminal du serveur.

Gestion des Commandes Client

Le serveur traite trois commandes principales (G, O, S) émises par les clients durant le jeu :

- **Commande G (Accusation)** :

```
    switch (buffer[0]) {
        case 'G': {
    int pid, guess;
    sscanf(buffer, "G %d %d", &pid, &guess);
    printf("DEBUG srv: joueur %d accuse %d\n", pid, guess);

    if (guess == culprit) {
        // 1) Envoi du message de victoire à tous
        sprintf(reply, "W %d %d", pid, culprit);
        broadcastMessage(reply);
        printf("DEBUG srv: joueur %d a gagné ! coupable = %s\n",
               pid, nomcartes[culprit]);

        // 2) Fermeture du serveur
        close(newsockfd);
        close(sockfd);
        exit(0);
    } else {
        // accusation incorrecte : ancien comportement
        broadcastMessage(buffer);
        joueurCourant = (joueurCourant + 1) % 4;
        sprintf(reply, "M %d", joueurCourant);
        broadcastMessage(reply);
        printf("DEBUG srv: passage au joueur %d\n", joueurCourant);
    }
    break;
}
```

- Si l'accusation est correcte (le suspect accusé est le coupable), le serveur annonce la victoire à tous et arrête la partie.
- Si l'accusation est incorrecte, le serveur annonce l'accusation aux joueurs et passe au joueur suivant.

- **Commande O (Exclusion d'objet)** :

```
        case 'O': {
    // 1) Parse
    int pid, col;
    sscanf(buffer, "O %d %d", &pid, &col);
    printf("DEBUG: exclusion objet %d par joueur %d\n", col, pid);

    // 2) Met à 0 la colonne sur tous les joueurs
    for (int k = 0; k < 4; k++) {
        tableCartes[k][col] = 0;
    }

    // 3) Envoie V… à tous
    for (int k = 0; k < 4; k++) {
        sprintf(reply, "V %d %d %d", k, col, tableCartes[k][col]);
        broadcastMessage(reply);
        printf("DEBUG: envoi %s\n", reply);
    }

    // 4) Passe au joueur suivant
    joueurCourant = (joueurCourant + 1) % 4;
    sprintf(reply, "M %d", joueurCourant);
    broadcastMessage(reply);
    printf("DEBUG: nouveau tour %d\n", joueurCourant);
    break;
}
```

- Le serveur met à jour la grille tableCartes en excluant un objet pour tous les joueurs.
- Diffuse la mise à jour à tous les clients.
- Passe au joueur suivant.

- **Commande S (Question à un joueur)** :

```
        case 'S': {
    // Question : S <pid> <targetPlayer> <col>
    int pid, target, col;
    sscanf(buffer, "S %d %d %d", &pid, &target, &col);
    printf("DEBUG: joueur %d questionne joueur %d sur objet %d\n",
           pid, target, col);

    // Logique d'exemple : on incrémente la valeur (mod 3)
    tableCartes[target][col] = (tableCartes[target][col] + 1) % 3;

    // On envoie la mise à jour à tous
    sprintf(reply, "V %d %d %d", target, col, tableCartes[target][col]);
    broadcastMessage(reply);
    printf("DEBUG: broadcast %s\n", reply);

    // Passage au joueur suivant
    joueurCourant = (joueurCourant + 1) % 4;
    sprintf(reply, "M %d", joueurCourant);
    broadcastMessage(reply);
    printf("DEBUG: broadcast %s\n", reply);

    break;
}
```

  - Le serveur modifie la valeur associée à l'objet demandé pour le joueur interrogé (modulo 3).
  - Diffuse la mise à jour à tous les clients.
  - Passe au joueur suivant.

c. Aspects Techniques Utilisés

- **Sockets TCP** :

Le serveur utilise des sockets TCP POSIX pour gérer la communication avec les clients. Grâce aux fonctions standards telles que socket(), bind(), listen() et accept(), le serveur attend et accepte les connexions entrantes des quatre joueurs. Chaque connexion établie permet de récupérer des informations essentielles sur chaque joueur (adresse IP, port, nom d'utilisateur) qui sont stockées dans une structure dédiée (tcpClients). Chaque joueur reçoit un identifiant unique allant de 0 à 3, et une fois les quatre joueurs connectés, le serveur diffuse ces informations à tous pour garantir une parfaite synchronisation initiale.

III. Client

a. Rôle du Client

Le client est chargé d'afficher graphiquement l'état du jeu (cartes, objets, suspects), de permettre l'interaction utilisateur et de transmettre les commandes correspondantes au serveur.

b. Code Complété et Explications

Envoi des commandes (G, O, S)

Les actions des joueurs sont envoyées au serveur via des commandes TCP :

```
    sprintf(sendBuffer, "G %d %d", gId, guiltSel);
    // Envoi de la commande « G » au serveur
    sendMessageToServer(gServerIpAddress, gServerPort, sendBuffer);
}
else if ((objetSel != -1) && (joueurSel == -1))
{
    sprintf(sendBuffer, "O %d %d", gId, objetSel);
    // Envoi de la commande « O » au serveur
    sendMessageToServer(gServerIpAddress, gServerPort, sendBuffer);
}
else if ((objetSel != -1) && (joueurSel != -1))
{
    sprintf(sendBuffer, "S %d %d %d", gId, joueurSel, objetSel);
    // Envoi de la commande « S » au serveur
    sendMessageToServer(gServerIpAddress, gServerPort, sendBuffer);
}
```

- **Accusation (G)** : Envoie une accusation sur un suspect.
- **Exclusion (O)** : Exclut un objet pour tous.
- **Question (S)** : Pose une question sur un objet à un autre joueur.

Traitement des Réponses Serveur

Le client traite plusieurs messages du serveur pour actualiser l'interface utilisateur :

- **Message I** :

  Le serveur attribue un identifiant unique au client. Cette information est utilisée pour déterminer si le joueur est actif pendant son tour (comparaison avec le joueur courant).

```
    case 'I':
        // Le serveur m'attribue mon ID
        sscanf(gbuffer, "I %d", &gId);
        printf("Mon ID = %d\n", gId);
        break;
```

- **Message L** :
  Contient les noms des quatre joueurs connectés. Elle est stockée dans un tableau gNames[] pour affichage dans l'interface.

```
case 'L':
    // Le serveur envoie la liste des joueurs
    {
        char n0[40], n1[40], n2[40], n3[40];
        sscanf(gbuffer, "L %39s %39s %39s %39s", n0, n1, n2, n3);
        strcpy(gNames[0], n0);
        strcpy(gNames[1], n1);
        strcpy(gNames[2], n2);
        strcpy(gNames[3], n3);
    }
    break;
```

- **Message D** :
  Le serveur envoie les trois cartes secrètes du joueur. Ces cartes sont affichées sur l'interface graphique dans la zone dédiée au joueur courant.

```
case 'D':
    // Le serveur m'envoie mes 3 cartes
    {
        int c0, c1, c2;
        sscanf(gbuffer, "D %d %d %d", &c0, &c1, &c2);
        b[0] = c0;
        b[1] = c1;
        b[2] = c2;
    }
    break;
```

- **Message M** :
  Indique quel joueur peut effectuer une action.
  Si l'ID reçu correspond à celui du client, le bouton "Go" devient actif (goEnabled = 1), sinon il est désactivé.

```
case 'M':
    // Le serveur indique le joueur courant
    {
        int current;
        sscanf(gbuffer, "M %d", &current);
        goEnabled = (current == gId);
    }
    break;
```

- **Message V** :
  Met à jour la table tableCartes[][] qui reflète les objets visibles et supposés. Ces valeurs sont directement utilisées pour afficher la grille de déduction.

```
case 'V':
    // Le serveur met à jour une valeur de tableCartes
    {
        int pj, col, val;
        sscanf(gbuffer, "V %d %d %d", &pj, &col, &val);
        tableCartes[pj][col] = val;
    }
    break;
```

c. Aspects Techniques Utilisés

- **Sockets TCP** : e client utilise des sockets TCP POSIX pour établir une connexion fiable avec le serveur. Il récupère les informations du serveur, configure l'adresse du serveur, établit la connexion, envoie des données puis ferme le socket après chaque envoi, assurant une communication stable et ordonnée avec le serveur tout au long du jeu.
- **Thread** :

Le client utilise un thread pour gérer en parallèle la réception continue des messages du serveur afin de ne pas bloquer l'interface utilisateur graphique.

- **Mutex** :

Le mutex est prévu pour sécuriser l'accès à la mémoire partagée (gbuffer) afin d'éviter des problèmes potentiels d'accès concurrents.

POLYTECH
SORBONNE