ISSAAD Samir EISE4
BASSECOURT Marc

Rapport SH13 - OSUSER

Commençons dans un premier temps par détailler le but du projet qui est basé sur un jeu de société du nom de Sherlock 13 qui se joue à 4 joueurs, le principe est le suivant :
Le jeu est composé de 13 cartes personnages. 3 cartes sont distribuées à chacun des 4 joueurs et la dernière est mise de côté, celle-ci est donc la carte représentant le coupable de cette partie. Le but des joueurs est de trouver le nom de ce personnage et de lancer une accusation.
Pour cela chaque joueur possède des symboles sur chacune de ses cartes comme on peut ci-dessous :

Il existe 8 types de symboles différents et chaque carte personnage contient des symboles différents comme on peut le voir ci-dessous

Le joueur va donc pouvoir réaliser différentes actions qui vont lui permettre de déterminer les cartes possédées par chacun des autres joueurs et donc de déterminer la carte coupable.
3 actions sont donc possibles :

- Demander le nombre précis d'un certain type de symbole à un joueur (exemple : joueur 4 combien possèdes-tu de livres ?)
- Demander à tous les joueurs si ils possèdent un certain type de symbole (exemple : possédez-vous des livres ?)
- Lancer une accusation sur un personnage. Dans ce cas, si l'accusation est bonne, le joueur remporte la partie.

Nous avions pour cela à notre disposition 2 fichiers

- *server.c*
  Un fichier gérant le côté du serveur qui permettra aux 4 joueurs désirant lancer une partie de se connecter à celui-ci et qui permettra par la suite de gérer l'envoi et la réception des données de chaque joueur afin de pouvoir traiter toutes les informations et déterminer la suite de la partie.
- *sh13.c*
  Un fichier gérant le côté du joueur et permettant à ce dernier de lancer une partie en se connectant à un serveur, une fois les 4 joueurs connectés ce dernier recevra donc un ensemble de cartes de la part du serveur et à partir de ce moment, la partie commence et le joueur pourra donc réaliser l'une des actions décrites ci-dessus à chaque tour afin d'avancer dans le jeu, chaque action étant transmise au serveur dans le but qu'il traite les informations et retourne au joueur la réponse attendue.

Pour lancer une partie, il suffit donc de lancer les 2 fichiers .sh fournis avec le code

- cmd.sh

```
#! /bin/sh
gcc -o sh13 -I/usr/include/SDL2 sh13.c -lSDL2_image -lSDL2_ttf -lSDL2 -lpthread
gcc -o server server.c
./server 32000
```

Ce fichier permet de réaliser la compilation des 2 fichiers .c précédents et de lancer le serveur avec un numéro de port que l'on spécifie (ici 32000).

- jeu.sh

```
#! /bin/sh
./sh13 localhost 32000 localhost 32001 Marc &
./sh13 localhost 32000 localhost 32002 Samir &
./sh13 localhost 32000 localhost 32003 Arthur &
./sh13 localhost 32000 localhost 32004 Ludo &
```

Ce fichier permet de lancer 4 interfaces joueurs directement en spécifiant dans un premier temps l'adresse du serveur, puis son port avant de spécifier l'adresse du client (joueur) et son port qui doit donc être différent de celui du serveur et des autres joueurs si l'adresse ip est identique, ce qui est le cas ici étant donnée que les tests ont été réalisés sur un réseau local, d'où l'utilisation de "localhost".

Lorsqu'un client souhaite lancer une partie il établit donc une connexion avec le serveur via un socket qui permettra la communication entre ces 2 entités. Lors de l'établissement de la connexion, le client transmet donc son adresse ip, son numéro de port ainsi que le nom du joueur afin que les 2 entités puissent continuer à communiquer par la suite. Le serveur crée donc un nouveau socket en acceptant la demande du client et sauvegarde les informations transmises par chaque client. Une fois que le serveur a établi 4 connexions avec 4 joueurs la partie peut donc commencer et plus aucune connexion ne sera possible notamment grâce au listen(5) qui ne permet d'accepter qu'un total de 5 demandes mais également car la partie est lancée.

Une fois la partie lancée, il s'ensuit un échange de communication entre le serveur et ses clients (joueurs) afin de faire avancer la partie. Pour cela chacun des fichiers sh13.c et server.c possèdent respectivement 2 fonctions sendMessageToServer et sendMessageToClient.

```c
void sendMessageToClient(char *clientip,int clientport,char *mess)
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    server = gethostbyname(clientip);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(clientport);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        {
                printf("ERROR connecting\n");
                exit(1);
        }

        sprintf(buffer,"%s\n",mess);
        n = write(sockfd,buffer,strlen(buffer));

    close(sockfd);
}
```

Le principe de ces 2 fonctions est identique, elles prennent en paramètre une adresse ip, un numéro de port ainsi qu'un message et transmettent l'information via un write sur le socket concerné.

La fonction broadcast quant à elle se trouve uniquement dans le server.c et permet de transmettre le même message à tous les joueurs actuellement en jeu via un simple boucle qui reprend la fonction sendMessageToClient.

```
void broadcastMessage(char *mess)
{
        int i;

        for (i=0;i<nbClients;i++)
                sendMessageToClient(tcpClients[i].ipAddress,
                        tcpClients[i].port,
                        mess);
}
```

Les messages transmis entre le serveur et ses clients sont composés de cette manière : “COM paramètre1 paramètre2 ....”. COM représente donc un caractère clé qui permet d'identifier le type d’informations envoyé ou reçue (Accusation, demande de symboles…), ce qui permet de réaliser différentes actions avec les paramètres fournis.

Le fichier sh13.c quant à lui doit gérer plusieurs choses étant donnée qu’il faut également gérer l’affichage graphique pour l’interface du joueur. Cette affichage est réalisé à l’aide de la bibliothèque SDL. Le fait que le client gère également un affichage graphique implique l’utilisation d’un thread étant donnée que le joueur doit s’assurer de la bonne réception des messages envoyés par le serveur tout en conservant un affichage graphique fonctionnel.

```
synchro=0;
ret = pthread_create ( & thread_serveur_tcp_id, NULL, fn_serveur_tcp, NULL);
```

Grâce au thread crée ci-dessus, le client peut donc écouter le serveur en continu sans interrompre ou rendre indisponible l’affichage graphique. Si le client reçoit des informations la variable synchro passe à 1 afin d’informer que des informations doivent être mise à jour sur l’interface graphique.

Pour finir, nous avons décidé d’ajouter quelques améliorations afin de bien comprendre le fonctionnement de la bibliothéque SDL ainsi que la communication dans un réseau. Nous avons pour cela rajouter un écran de fin qui est géré par un nouveau message envoyé à tous les joueurs par le serveur afin afin d’indiquer la fin de partie à tous les joueurs ainsi que l’issue de la partie avec le nom du joueur gagnant et le nom du coupable de cette partie.