Sorbonne Universités - Ecole Polytechnique Universitaire

Système d'exploitation

Rapport TP Système d'exploitation

Authors :
ASMA Samy
LACHHAB Ibtissam

Student Numbers :
3670516
3800650

4 juin 2019

Table des matières

1 Introduction

Dans le cadre de notre formation MAIN, nous avons, au deuxième semestre de première année, un cours de système d'exploitation. Ce cours est divisé en cours en salle avec un professeur, et des cours de travaux pratiques. Ce rapport est destiné à résumer et expliciter notre projet.
Le but de ce dernier est d'implémenter le jeu Sherlock 13 en réseau. Dans ce jeu, chaque joueur possède 3 cartes, qui chacune possède 3 objets. Chaque joueur a le rôle du détective, son but est de découvrir quelle carte n'a pas été distribuée et découvrir qui est le coupable, tout cela en interrogeant les autres joueurs afin de découvrir leur cartes.
Nous allons premièrement donner une explication des outils pour faire un jeu réseau, puis faire une étude détaillée du code et enfin présenter le jeu. Le code sera écrit en C.

2 Partie A : Les outils indispensables

2.1 Les sockets

Qu'est ce qu'une Socket ?
Une socket est un élément logiciel qui est aujourd'hui répandue dans la plupart des systèmes d'exploitation. Il s'agit d'une interface logicielle avec les services du système d'exploitation, grâce à laquelle un développeur exploitera facilement et de manière uniforme les services d'un protocole réseau. Il lui sera ainsi par exemple aisé d'établir une session TCP, puis de recevoir et d'expédier des données grâce à elle.

2.2 Les pthreads

Qu'est ce qu'un Thread ?
Un thread est une portion de code (fonction) qui se déroule en parallèle du thread principal (aussi appelé main). Ce principe est un peu semblable à la fonction fork sur Linux par exemple sauf que nous ne faisons pas de copie du processus père, nous définissons des fonctions qui vont se lancer en même temps que le processus, ce qui permet de faire de la programmation multitâche. Le but est donc de permettre au programme de réaliser plusieurs actions au même moment (imaginez un programme qui fait un gros calcul et une barre de progression qui avance en même temps).

Nous l'utiliserons dans notre projet afin d'effectuer plusieurs tâches côté client, que nous verrons à la suite de ce rapport.

2.3 SDL

Qu'est-ce que le SDL ?
La SDL c'est-à-dire la Simple DirectMedia Layer est une bibliothèque multimédia. Elle permet un accès de bas-niveau à l'audio, au clavier, à la souris, au joystick, aux graphiques... Cela veut dire qu'elle permet d'afficher des fenêtres, d'afficher des images, de jouer des sons, de gérer le clavier.

Il s'agit d'une bibliothèque libre, multiplateforme et assez connue.

3 Partie B : Explication du code

Afin de réaliser notre projet, nous avons besoin d'écrire deux codes qui seront intimement liés, celui du serveur qui regroupera toutes les informations sur le jeu et qui traitera les informations envoyées par les joueurs, et celui du joueur qui enverra toutes les informations sur le joueur aux serveurs et les différentes actions qu'il effectue.

3.1 Transmission Control Protocol

Dans le mode connecté (théoriquement : instauration d'une session de communication entre plusieurs parties requiérant l'échange de données), TCP est un protocole de transmission fiable.

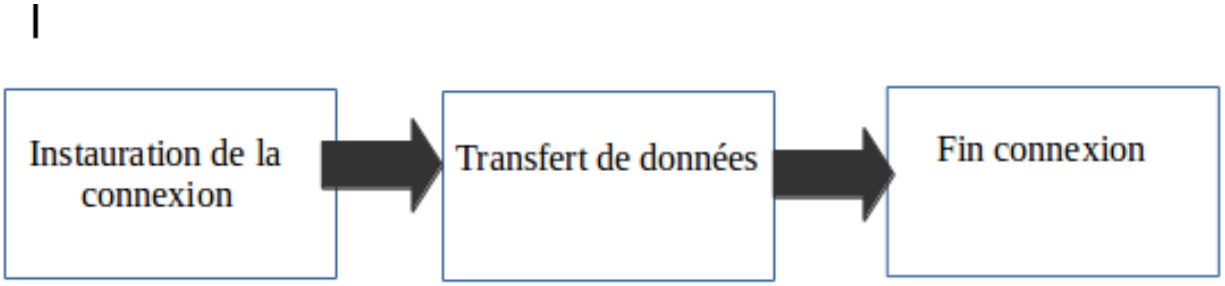

FIGURE 1 – Fonctionnement TCP

La machine émettrice (demandeuse de connexion) est le **client**, la machine réceptrice est le **server**. TCP est appelé fiable car il permet d'assurer entres autres la bonne réception des paquets de données (ajout d'un en-tête précédant les paquets de données) et l'ordonnancement des segments (afin de recevoir dans l'ordre souhaité le message).

3.2 Code Client

3.2.1 Glossaire des variables

gServerIpAddress [256] [char] : stocke l'adresse IP du serveur

gbuffer[256] [char] : récupère ce qui est entré dans le terminal

gServerPort [int] : stocke le numéro de port du serveur

gClientIpAddress[256][char] : stocke l'adresse IP du client

gClientPort[int] : stocke le numéro de port du client

gName[256][char] : nom du client

gNames[4][256][char] : nom de l'ensemble des joueurs

gId[int] : identifiant du client

joueurSel[int] : joueur sélectionné

objetSel[int] : objet sélectionné

guiltSel[int] : coupable sélectionné

guiltGuess[13][int] : ensemble des coupables possibles

tableCartes[4][8][int] : stocke les objets de chaque joueur

b[3][int] : stocke les 3 cartes du joueur

goEnabled[int] :

connectEnabled[int] :

chiffre[int] :

***nbobjets**[] : nombre de chaque objet

***nbnoms**[] : nombre de l'ensemble des cartes

synchro[volatile int] : variable dont la valeur peut être modifiée à l'extérieur du programme.

3.2.2 Fonctions

```
1 void *fn_serveur_tcp(void *arg)
```

Création de la socket
On déclare la socket avec le type $SOCKET$; puis on la crée à l'aide de la fonction $socket(AF\_INET, SOCK\_STREAM, 0)$
Les arguments de la fonction sont expliqués par le fait que notre protocole utilisé est le TCP/IP. On se sert ensuite de la structure $sockaddr\_in$ pour configurer la connexion. On paramètre nos structures afin d'avoir l'IP donnée au serveur, afin d'en automatiser la recherche ; on se sert de la fonction $htons()$, et d'avoir également le port utilisé par le serveur. On établit, dans un troisième temps, une connexion avec le client. On joint les informations récoltées à la socket. On prévoit tous les cas d'erreurs. Puis on accepte la socket et on l'enregistre dans gbuffer. Enfin, lorsqu'un *thread* est lancé, une connection est faite et qu'il y a réception d'un message, *synchro* est égal à 1. On réitère le processus tant que synchro est non nul.

```
1 void sendMessageToServer(char *ipAddress, int portno, char *mess)
```

La fonction prend en paramètre l'adresse IP du client, son numéro de port, et le message à envoyer. A l'aide $htons()$, elle récupère automatiquement l'adresse IP et le numéro de port du serveur. Ainsi, elle peut créer une socket à l'aide des structures pré-définies, et y intégrer l'ensemble des informations nécessaires à l'envoi du message au serveur.

3.2.3 Main

```
1 int main(int argc, char ** argv)
```

On entre dans le *main*,

```
1 printf(<app> <Main server ip address> <Main server port> <Client ip address> <Client
      port> <player name>);
```

Pour la compilation, on demande à l'utilisateur 5 entrées qu'on copiera dans nos variables. Ainsi $gServerIpAddress$ prend l'argument 1, $gServerPort$ est converti en entier par la fonction *atoi* et prend le deuxième argument ; et ainsi de suite pour $gClientIpAddress$, $gClientPort$ et $gName$. Par exemple, un joueur qui souhaiterait se connecter devra rentrer :
$./sh13$ *localhost* 32000 *localhost* 32001 *Samy*

On crée une fenêtre à l'aide de la fonction $CreateWindow$ de taille 1024×768, on prévoit de découper l'espace de la fenêtre entre l'espace pour les objets, l'espace pour le deck, l'espace pour le bouton *go* et l'espace pour le bouton *connect*.
A l'aide de la fonction IMG_Load on charge les images des cartes, des objets ainsi des boutons

go et *connect*.
On copie tous les noms des joueurs entrés par eux-mêmes et on rentre l'ensemble dans $gNames$.

On initialise nos variables :

```
joueurSel=-1;
objetSel=-1;
guiltSel=-1;

b[0]=-1;
b[1]=-1;
b[2]=-1;
```

Puis tant que le joueur ne quitte pas le jeu $quit \neq 0$, on va agir sur nos variables. Pour commencer, on va contrôler la souris, en fonction de sa position sur l'interface ($SDL\_MOUSEBUTTONDOWN$)

- Lorsque le joueur va cliquer sur le bouton *connect*, le message commençant par la lettre C est envoyé au serveur, ainsi le client va pouvoir se connecter.
- Changement de l'état des variables $joueurSel$, $guiltSel$ et $objetSel$ si sélectionnés par le joueur.

```
else if ((mx>=0) && (mx<200) && (my>=90) && (my<330))
{
        joueurSel=(my-90)/60;
        guiltSel=-1;
}
else if ((mx>=200) && (mx<680) && (my>=0) && (my<90))
{
        objetSel=(mx-200)/60;
        guiltSel=-1;
}
else if ((mx>=100) && (mx<250) && (my>=350) && (my<740))
{
        joueurSel=-1;
        objetSel=-1;
        guiltSel=(my-350)/30;
}
else if ((mx>=250) && (mx<300) && (my>=350) && (my<740))
{
        int ind=(my-350)/30;
        guiltGuess[ind]=1-guiltGuess[ind];
}
```

- Lorsque le joueur sélectionne un coupable, on envoie un message au serveur commençant par la lettre G

```
if (guiltSel!=-1)
{
        sprintf(sendBuffer,G %d %d,gId, guiltSel);
        sendMessageToServer(gServerIpAddress,gServerPort,sendBuffer);

}
```

— Lorsque le joueur sélectionne un objet, et désire savoir quels joueurs possèdent cet objet, on envoie un message au serveur commençant par la lettre O

```
else if ((objetSel!=-1) && (joueurSel==-1))
{
        sprintf(sendBuffer,O %d %d,gId, objetSel);
        sendMessageToServer(gServerIpAddress,gServerPort,sendBuffer);
}
```

— Si le joueur sélectionne un objet et un joueur, afin de savoir le nombre d'occurences de l'objet chez le joueur sélectionné, on envoie un message au serveur commençant par la lettre S

```
else if ((objetSel!=-1) && (joueurSel!=-1))
{
        sprintf(sendBuffer,S %d %d %d,gId, joueurSel,objetSel);
}
```

Chaque message envoyé par le serveur sera traité à partir de la première lettre contenue dans le buffer. Ainsi on obtient :

Cas `I` : le joueur reçoit son identifiant, on initialise donc gId.

```
case 'I':
        gId= (int) gbuffer[2] -(int)('0');
        break;
```

Cas `L` : On va stocker l'ensemble des noms des joueurs dans $gNames$, le joueur va en recevoir la liste.

```
case 'L':
        sscanf(gbuffer,L %s %s %s %s,gNames[0],gNames[1],gNames[2],gNames[3]);
        break;
```

Cas `D` : Le joueur va recevoir ses trois cartes distribuées. On va stocker leurs numéros dans b.

```
case 'D':
        sscanf(gbuffer,D %d %d %d,&b[0],&b[1],&b[2]);
        break;
```

Cas `M` : Le joueur va recevoir le numéro du joueur qui joue en premier. Le bouton *go* va s'activer pour le premier joueur.

```
case 'M':
        chiffre=-1;
        sscanf(gbuffer,M %d,&chiffre);
        if (chiffre==gId)
        {
                goEnabled=1;
        }
        else
        {
                goEnabled=0;
        }
        break;
```

Cas `V` : On va enregistrer la valeur que le joueur reçoit dans son tableau de cartes.

```
case 'V':
        sscanf(gbuffer,V %d %d %d,&a1,&a2,&val);
        tableCartes[a1][a2]=val;
        break;
```

3.3 Code serveur

Ici nous allons expliquer le code du serveur et faire le lien avec la partie précedente

3.3.1 Les Variables

Listing 1 – Variable Serveur

```
struct _client
{
        char ipAddress[40];
        int port;
        char name[40];
        int etat;
} tcpClients[4];

int nbClients;
int fsmServer;

int deck[13]={0,1,2,3,4,5,6,7,8,9,10,11,12};

int tableCartes[4][8];

char *nomcartes[]=
{Sebastian Moran, irene Adler, inspector Lestrade,
  inspector Gregson, inspector Baynes, inspector Bradstreet,
  inspector Hopkins, Sherlock Holmes, John Watson, Mycroft Holmes,
  Mrs. Hudson, Mary Morstan, James Moriarty};

int joueurCourant;
```

La structure tcpClients est un tableau pour stocker les informations sur les joueurs, on stocke leurs adresses ip, numéros de port, noms et états.
nbClient permet de savoir le nombre de clients connectés, $fsmServer$ permet de savoir si les quatre joueurs sont bien connectés.
deck[13] stocke les numéros de chaque carte du deck. *tablesCartes* permet de savoir le nombre de chaque objet que possède chaque joueur *nomcartes*[] stocke le nom de chaque carte $joueurCourant$ stocke l'id du joueur courant

3.3.2 Les fonctions

Listing 2 – Fonction Serveur

```
void error(const char *msg);
void melangerDeck();
void createTable();
void printDeck();
void printClients();
int findClientByName(char *name);
void sendMessageToClient(char *clientip,int clientport,char *mess);
void broadcastMessage(char *mess);
```

La fonction error, permet de lancer un message d'erreur avec un message personnalisé.
MelangerDeck va nous permettre de mélanger le deck grâce à la fonction aléatoire.
createTable va distribuer trois cartes à chaque joueur et leur attribuer les objets correspondants.
printDeck va afficher les cartes ainsi que la table de cartes.
printClient va afficher l'adresse ip, le numéro de port et le nom de chaque client.
findClientByName va trouver l'indice du client dans tcpClient grâce au nom du client.
sendMessageToClient va envoyer un message au client donc l'adresse ip et le numéro de port sera donné en argument broadcastMessage va envoyer un message à tout les clients connectés.

3.3.3 Le Main

Listing 3 – Variable Serveur

```
int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno;
     socklen_t clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
         int i;

        char com;
        char clientIpAddress[256], clientName[256];
        int clientPort;
        int id;
        char reply[256];


     if (argc < 2) {
         fprintf(stderr,ERROR, no port provided);
         exit(1);
```

```
20      }
21      sockfd = socket(AF_INET, SOCK_STREAM, 0);
22      if (sockfd < 0)
23         error(ERROR opening socket);
24      bzero((char *) &serv_addr, sizeof(serv_addr));
25      portno = atoi(argv[1]);
26      serv_addr.sin_family = AF_INET;
27      serv_addr.sin_addr.s_addr = INADDR_ANY;
28      serv_addr.sin_port = htons(portno);
29      if (bind(sockfd, (struct sockaddr *) &serv_addr,
30               sizeof(serv_addr)) < 0)
31               error(ERROR on binding);
32      listen(sockfd,5);
33      clilen = sizeof(cli_addr);
34
35         printDeck();
36         melangerDeck();
37         createTable();
38         printDeck();
39         joueurCourant=0;
40
41         for (i=0;i<4;i++)
42         {
43                 strcpy(tcpClients[i].ipAddress,localhost);
44                 tcpClients[i].port=-1;
45                 tcpClients[i].etat=1;
46                 strcpy(tcpClients[i].name,-);
47         }
```

On va d'abord voir l'initialisation du jeu dans le main, on initialise plusieurs variables locales comme $buffer$ et $reply$ qui vont nous servir à stocker du texte ainsi que des structures de socket pour recevoir le message par le protocole tcp On initialise la socket, on met l'adresse ip du serveur et son numéro de port. On affiche le deck non mélangé puis on mélange le deck, on crée la table puis on réaffiche le deck. On initialise le joueur courant à 0 (premier joueur). On initialise $tcpCLient$.

Dans le $while(1)$, on accepte la socket qui arrive, Si fmsServer=0, on lit les message qui arrivent. S'ils commencent par `C` on extrait les informations nécessaires telles que le message, adresse ip et numéro de port du client. On renvoie un message au client lui indiquant son Id avec le message commençant par `I` et on incrémente de 1 le nombre de clients. Lorsque le nombre de clients atteint 4 on distribue les cartes en utilisant comme première lettre `D` et les objets à chaque client et grâce à un on initialise $fsmserver$ à 1.

Lorsque fsmServer est à 1, cela signifie que la partie peut commencer. On recoit les informations sur les actions du client/joueur.

Si le message qui arrive commence par `G`, cela signifie que le client désigne un coupable. Si le coupable est bien la 12eme carte, le joueur a gagné et la partie s'arrête sinon il n'a plus le droit de jouer. Si le message qui arrive commence par `0` cela signifie que le joueur a sélectionné un objet et veut savoir quels sont les joueurs qui le possèdent, on vérifie donc dans *tableCarte* si la valeur est différente de 0, si oui on envoie la valeur 100, ce qui va permettre d'afficher un * dans le *tableCarte* du client.

Si le message qui arrive commence par `S`, c'est qu'il a sélectionné un joueur et un objet et veut savoir le nombre d'occurences de cet objet le joueur sélectionné possède, on regarde dans *tableCartes* et on envoie l'information à tous les joueurs.

Lorsque cette étape est finie, on passe au joueur suivant en regardant son état, si il est à 0, on passe au joueur suivant si non à celui d'après si son état est non nul !

4 Le jeu

Désormais nous allons voir le fonctionnement du jeu

— **Serveur au lancement :**

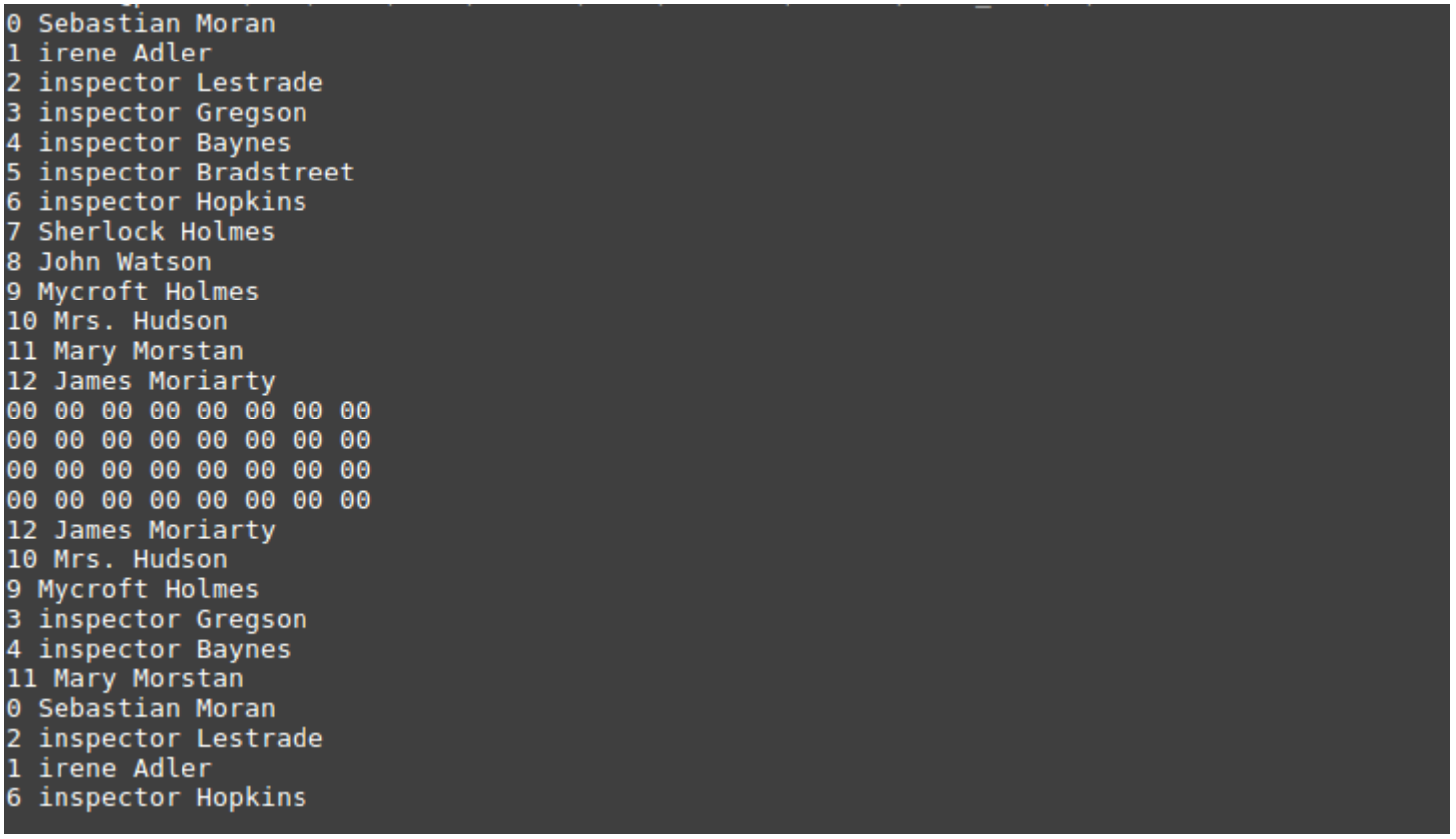

FIGURE 2 – Serveur au lancement

On voit bien qu'il y a un mélange des cartes, et que tableCarte est initialisée à 00.

— **Connexion du joueur 1**

	5	5	5	5	4	3	3	3
Samy								
-								
-								
-								

	Sebastian Moran	
	irene Adler	
	inspector Lestrade	
	inspector Gregson	
	inspector Baynes	
	inspector Bradstreet	
	inspector Hopkins	
	Sherlock Holmes	
	John Watson	
	Mycroft Holmes	
	Mrs. Hudson	
	Mary Morstan	
	James Moriarty	

FIGURE 3 – Connexion du joueur 1

	5	5	5	5	4	3	3	3
Samy								
Ibtissam								
Thomas								
Jacques	1	0	2	1	1	1	0	1

	Sebastian Moran	
	irene Adler	
	inspector Lestrade	
	inspector Gregson	
	inspector Baynes	
	inspector Bradstreet	
	inspector Hopkins	
	Sherlock Holmes	
	John Watson	
	Mycroft Holmes	
	Mrs. Hudson	
	Mary Morstan	
	James Moriarty	

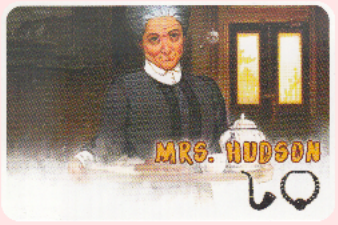

FIGURE 4 – Connexion du joueur 1

On voit bien que lorsque le joueur 1 est le seul connecté, le jeu ne peut commencer. Lorsque les 4 joueurs sont connectés, les cartes sont distribuées et la *tablesCartes* est remplie. Cette information est disponible pour tous les autres joueurs.

— **jeu du joueur 1**

	5	5	5	5	4	3	3	3
Samy	2	1	0	2	2	0	2	0
Ibtissam				0				
Thomas								
Jacques								

	Sebastian Moran	
	irene Adler	
	inspector Lestrade	
	inspector Gregson	
	inspector Baynes	
	inspector Bradstreet	
	inspector Hopkins	
	Sherlock Holmes	
	John Watson	
	Mycroft Holmes	
	Mrs. Hudson	
	Mary Morstan	
	James Moriarty	

FIGURE 5 – Jeu du joueur 1

Le joueur 1 (Samy) choisit un autre joueur et un objet, cela lui donne le nombre d'objets que possède Ibtissam.

— **jeu du joueur 2**

	5	5	5	5	4	3	3	3
Samy				*				
Ibtissam	2	1	2	0	1	1	1	0
Thomas				*				
Jacques				*				

	Sebastian Moran	
	irene Adler	
	inspector Lestrade	
	inspector Gregson	
	inspector Baynes	
	inspector Bradstreet	
	inspector Hopkins	
	Sherlock Holmes	
	John Watson	
	Mycroft Holmes	
	Mrs. Hudson	
	Mary Morstan	
	James Moriarty	

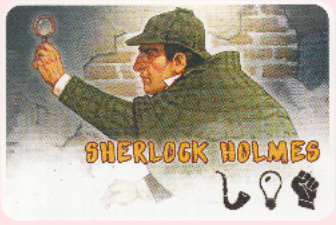

FIGURE 6 – Jeu du joueur 2

Le joueur 2 (Ibtissam) choisit un objet, cela lui donne toute les personnes qui possèdent cet objet en lui indiquant avec des astérisques. Cela signifie que la valeur dans *tableCartes* est de 100. Cette information s'affiche pour tous les autres joueurs.

— **jeu du joueur 3**

	5	5	5	5	4	3	3	3
Samy				*				
Ibtissam				0				
Thomas	0	2	1	2	0	0	0	1
Jacques				*				

Sebastian Moran	
irene Adler	
inspector Lestrade	
inspector Gregson	
inspector Baynes	
inspector Bradstreet	
inspector Hopkins	
Sherlock Holmes	
John Watson	
Mycroft Holmes	
Mrs. Hudson	
Mary Morstan	
James Moriarty	

FIGURE 7 – Jeu du joueur 3

Le joueur 3 (Thomas) choisit un coupable, malheureusement il se trompe, donc cela lui affiche Loser, et il ne pourra plus jouer.

— **jeu du joueur 4**

	5	5	5	5	4	3	3	3
Samy				*				
Ibtissam				0				
Thomas				*				
Jacques	1	0	2	1	1	1	0	1

Sebastian Moran	
irene Adler	
inspector Lestrade	
inspector Gregson	
inspector Baynes	
inspector Bradstreet	
inspector Hopkins	
Sherlock Holmes	
John Watson	
Mycroft Holmes	
Mrs. Hudson	
Mary Morstan	
James Moriarty	

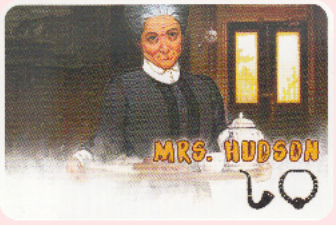

FIGURE 8 – Jeu du joueur 4

Le joueur 4 (Jacques) choisit un coupable, et il a raison !!! Cela lui affiche Winner, et il a gagné le jeu.

5 Conclusion

Ce cours de système d'exploitation nous a permis de découvrir le monde du réseau et les différentes applications à celui-ci, malgré la simplicité du jeu présenté, désormais nous avons toutes les connaissances de bases sur le TCP, socket et sur les pThread afin de pouvoir se perfectionner dans l'élaboration d'un jeu ou/et d'une application en réseau.