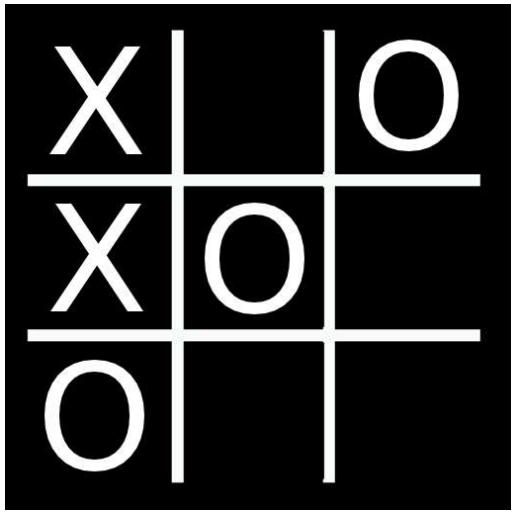
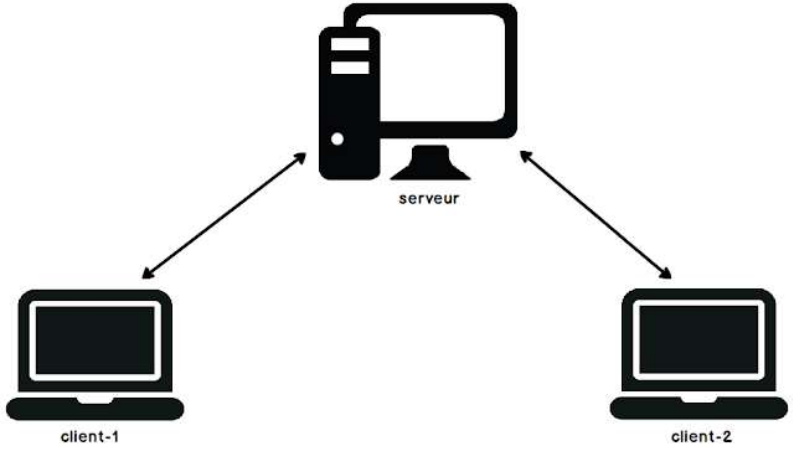
PROJET PSE - MORPION

PAR YASSER EL KOUHEN ET MARTIN HAMEL

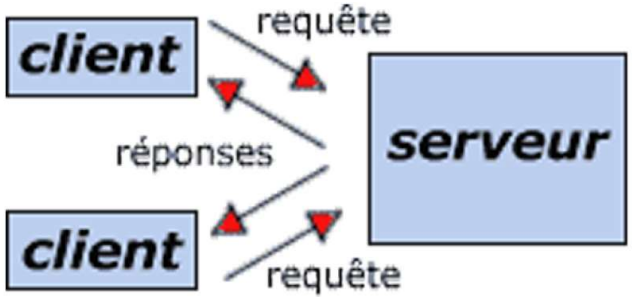
INTRODUCTION – LE JEU DU MORPION RÉSEAU



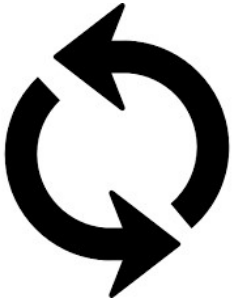


Réseau client-serveur

FONCTIONNALITÉS PRINCIPALES



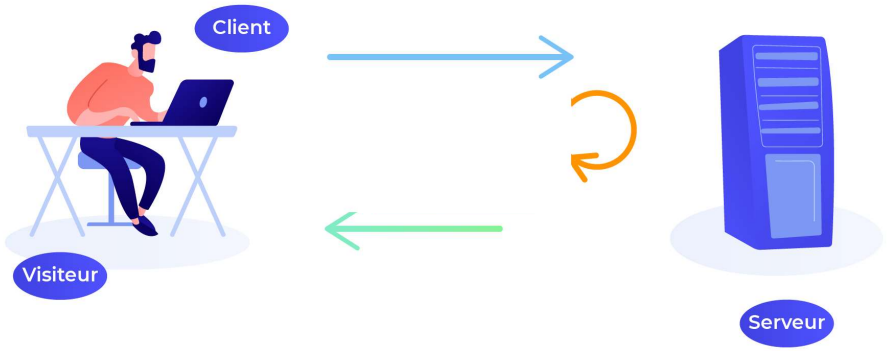
- Jeu en réseau pour 2 joueurs
- Interface utilisateur simple en ligne de commande



- Gestion des tours de jeu
- Détection des victoires et des matchs nuls

STRUCTURE DU CODE

- Envoi des mouvements
- Réception de la grille
- Interface utilisateur

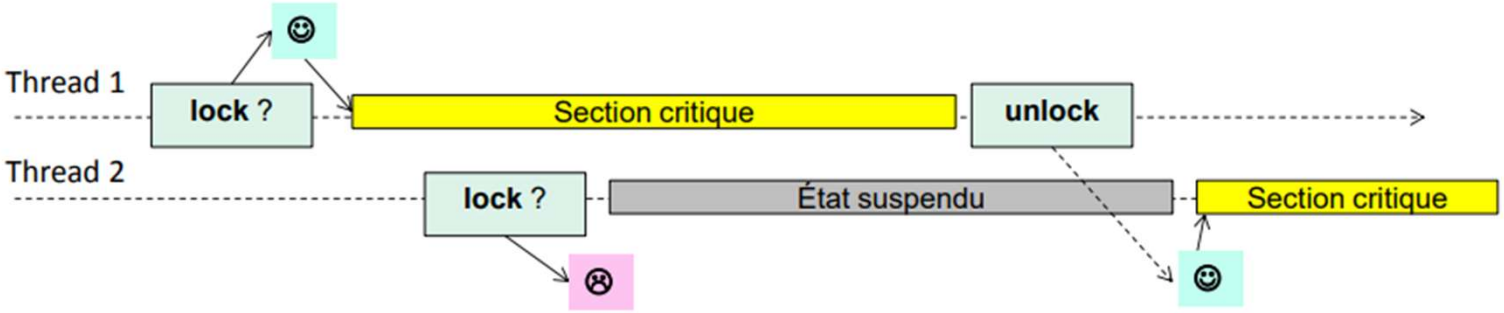


- Gestion des connexions
- Synchronisation
- Logique de jeu

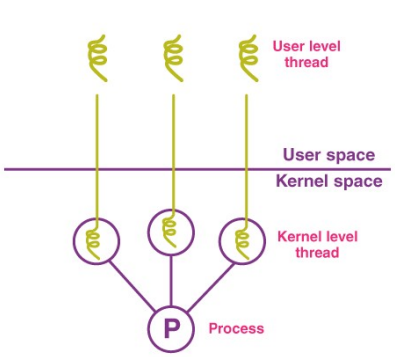
main() **handle_client()** **envoyer_grille()** **main()**

est_plein() **initialiser_grille()** **afficher_grille()** **verifier_gagnant()**

GESTION DE LA CONCURRENCE

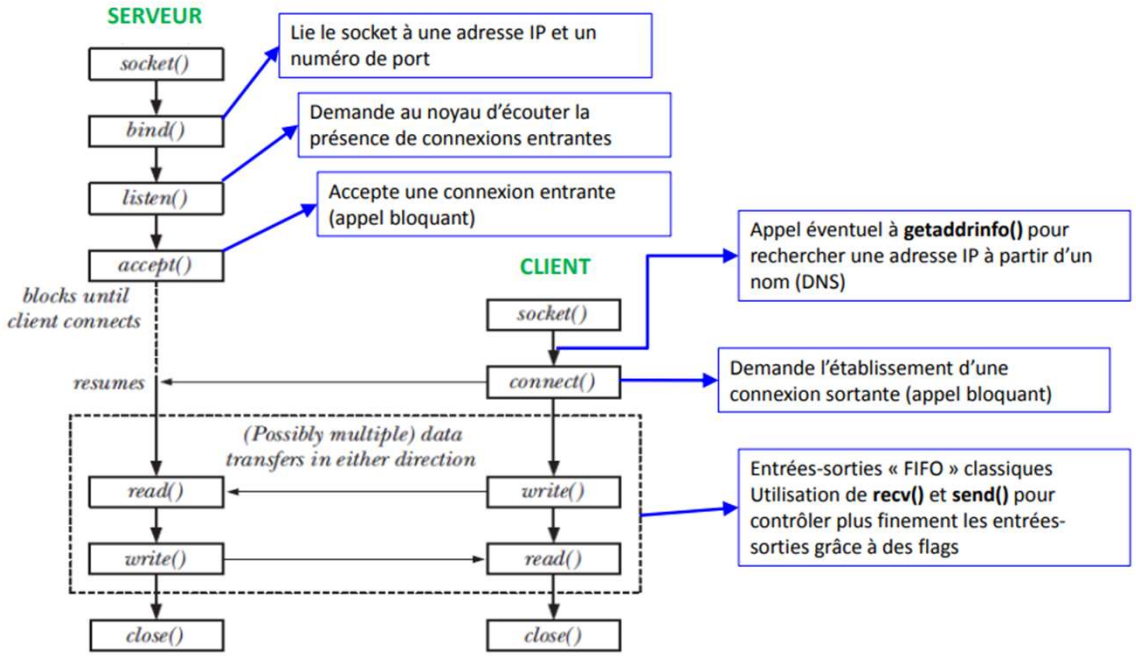


•**Synchronisation:** Mutex et variables conditionnelles pour assurer le tour par tour et l'intégrité des données



•**Threads**: Un thread par joueur pour gérer les connexions

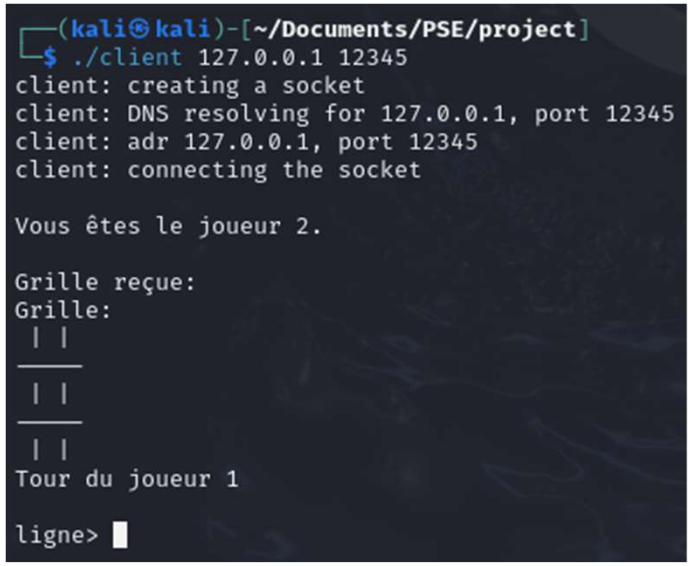
COMMUNICATION RÉSEAU



- **Sockets**: Utilisation des sockets pour la communication TCP entre le serveur et les clients

- **Messages**: Gestion des messages pour les mouvements des joueurs et les mises à jour de la grille

INTERFACE UTILISATEUR



•**Simplicité**: Interface en ligne de commande intuitive

•**Exemples**:

-Prompt pour les mouvements: ‘ ligne> ‘

-Affichage de la grille et des messages d'état

GESTION DES DÉCONNEXIONS

Serveur

```
┌──(kali㉿kali)-[~/Documents/PSE/proje
└─$ ./serveur
Serveur en écoute sur le port 12345
Joueur 1 connecté
Joueur 2 connecté
Client 1: 1 0
Client 2: 0 2
Client 1: 2 0
Client 2: 1 1
Client 1: fin

┌──(kali㉿kali)-[~/Documents/PSE/proje
└─$
```

Client 1

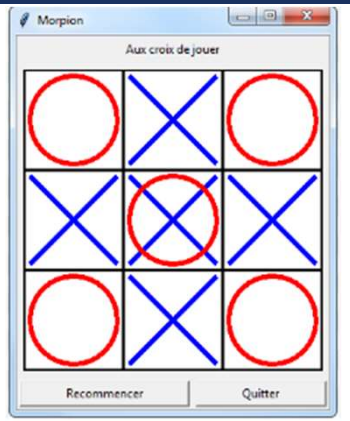
```
ligne> 2 0
client: 4 bytes sent
Grille reçue:
Grille:
 | |O
____
X| |
____
X| |
Tour du joueur 2

ligne> fin
client: 4 bytes sent

┌──(kali㉿kali)-[~/Documents/PSE/project]
└─$
```

Client 2

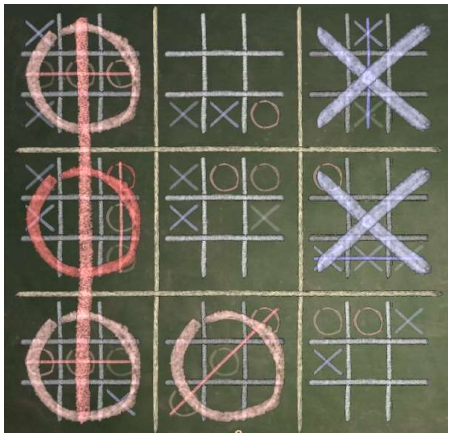
```
ligne> 1 1
client: 4 bytes sent
Grille reçue:
Grille:
 | |O
____
X|O|
____
X| |
Tour du joueur 1

ligne>
client: 1 bytes sent
Grille reçue:

Joueur 2 a gagné car l'adversaire s'est déconnecté de la partie !

ligne>

┌──(kali㉿kali)-[~/Documents/PSE/project]
└─$
```

- **Stratégie**: Détection des déconnexions, notification de l'autre joueur, fermeture des sockets
- **Code**: Exemple de gestion de déconnexion dans handle_client()

AMÉLIORATIONS POSSIBLES



- Ajout d'une interface graphique
- Support pour plus de deux joueurs (Métamorpion)
- Modes de jeu supplémentaires (ex: contre une IA)



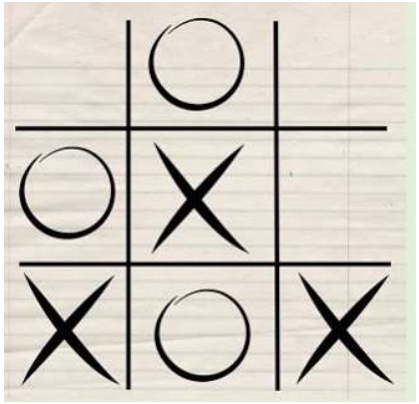
CONCLUSION

Résumé (points forts du projet)

•Gestion de la concurrence

•Interface utilisateur

•Robustesse



Apprentissage et Difficultés

•Gestion des Messages (taille et ordre)

•Gestion des déconnexions

• Exclusion Mutuelle

FIN… ET MERCI À VOUS !

ANNEXES

MORPION RÉSEAU

CODE SERVEUR 1

serveur.c

```c
#include "pse.h"

#define PORT 12345
#define NB_JOUEURS 2
#define TAILLE_GRILLE 3

typedef struct {
    int sock;
    int joueur;
} ClientInfo;

int joueurs_connectes = 0;
int fin_jeu = 0;
int joueur_actuel = 1;
char grille[TAILLE_GRILLE][TAILLE_GRILLE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

typedef struct {
    ClientInfo *clients;
    int index;
} ThreadData;

void *handle_client(void *arg);
void initialiser_grille();
void afficher_grille();
int verifier_gagnant();
int est_plein();
void envoyer_grille(int sock);

int main(int argc, char *argv[]) {
    int sock, client_sock, ret;
    struct sockaddr_in adrServ, adrClient;
    socklen_t lgAdrClient = sizeof(adrClient);
    pthread_t tid;
    ClientInfo clients[NB_JOUEURS];
    ThreadData data[NB_JOUEURS];

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }
```

client.c

```c

    adrServ.sin_family = AF_INET;
    adrServ.sin_addr.s_addr = INADDR_ANY;
    adrServ.sin_port = htons(PORT);

    if (bind(sock, (struct sockaddr *)&adrServ, sizeof(adrServ)) < 0) {
        perror("bind");
        exit(EXIT_FAILURE);
    }

    if (listen(sock, NB_JOUEURS) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    printf("Serveur en écoute sur le port %d\n", PORT);

    for (int i = 0; i < NB_JOUEURS; i++) {
        client_sock = accept(sock, (struct sockaddr *)&adrClient, &lgAdrClient);
        if (client_sock < 0) {
            perror("accept");
            exit(EXIT_FAILURE);
        }

        // Envoyer un message au client pour lui indiquer son numéro de joueur
        char msg[LIGNE_MAX];
        sprintf(msg, "Vous êtes le joueur %d.\n", i + 1);
        send(client_sock, msg, strlen(msg), 0);

        printf("Joueur %d connecté\n", i + 1);
        clients[i].sock = client_sock;
        clients[i].joueur = i + 1;
        data[i].clients = clients;
        data[i].index = i;

        ret = pthread_create(&tid, NULL, handle_client, (void *)&data[i]);
        if (ret ≠ 0) {
            perror("pthread_create");
            exit(EXIT_FAILURE);
        }

        pthread_mutex_lock(&mutex);
        joueurs_connectes++;
        if (joueurs_connectes == NB_JOUEURS) {
```

CODE SERVEUR 2

serveur.c × client.c ×

```
85          pthread_mutex_lock(&mutex);
86          joueurs_connectes++;
87          if (joueurs_connectes == NB_JOUEURS) {
88              pthread_cond_broadcast(&cond);
89          }
90          pthread_mutex_unlock(&mutex);
91      }
92
93      pthread_exit(NULL);
94      close(sock);
95      return 0;
96  }
97
98
99  void *handle_client(void *arg) {
100     ThreadData *data = (ThreadData *)arg;
101     ClientInfo *client = &(data->clients[data->index]);
102     ClientInfo *other_client = &(data->clients[1 - data->index]); // Ajout de cette ligne
103     char ligne[LIGNE_MAX];
104     int ret = 0;
105     int row, col;
106     char symbole = (client->joueur == 1) ? 'X' : 'O';
107     int vainqueur;
108
109     pthread_mutex_lock(&mutex);
110     while (joueurs_connectes < NB_JOUEURS) {
111         pthread_cond_wait(&cond, &mutex);
112     }
113     pthread_mutex_unlock(&mutex);
114
115     if (client->joueur == 1) {
116         initialiser_grille();
117     }
118
119     while (!fin_jeu) {
120         envoyer_grille(client->sock);
121
122         pthread_mutex_lock(&mutex);
123         while (joueur_actuel != client->joueur && !fin_jeu) {
124             pthread_cond_wait(&cond, &mutex);
125         }
126         pthread_mutex_unlock(&mutex);
127
128         if (fin_jeu) break;
```

serveur.c × client.c × Makefile

```
128         if (fin_jeu) break;
129
130         ret = recv(client->sock, ligne, LIGNE_MAX, 0);
131         if (ret < 0) {
132             perror("recv");
133             exit(EXIT_FAILURE);
134         } else if (ret == 0) {
135             printf("Client déconnecté\n");
136             fin_jeu = 1;
137             pthread_cond_broadcast(&cond);
138         } else if (ret==1) {
139             ligne[ret] = '\0';
140         } else {
141             ligne[ret] = '\0';
142             printf("Client %d: %s", client->joueur, ligne);
143
144             if (strcmp(ligne, "fin\n") == 0) {
145                 char msgf[LIGNE_MAX];
146                 if (client->joueur==1){
147                   vainqueur=2;
148                 } else {
149                   vainqueur=1;
150                 }
151                 sprintf(msgf, "\nJoueur %d a gagné car l'adversaire s'est déconnecté de la partie !
    \n",vainqueur);
152                 send(data->clients[0].sock, msgf, strlen(msgf), 0);
153                 send(data->clients[1].sock, msgf, strlen(msgf), 0);
154                 close(client->sock); // Ferme la socket du client actuel
155                 close(other_client->sock); // Ferme la socket du deuxième client
156                 fin_jeu = NB_JOUEURS;
157                 pthread_cond_broadcast(&cond);
158             } else {
159                 sscanf(ligne, "%d %d", &row, &col);
160                 pthread_mutex_lock(&mutex);
161                 if (grille[row][col] == ' ') {
162                     grille[row][col] = symbole;
163                     if (verifier_gagnant()) {
164                         envoyer_grille(client->sock);
165                         envoyer_grille(other_client->sock);
166                         char message[LIGNE_MAX];
167                         sprintf(message, "\nJoueur %d a gagné!\n", client->joueur);
168                         send(data->clients[0].sock, message, strlen(message), 0);
169                         send(data->clients[1].sock, message, strlen(message), 0);
170                         close(client->sock); // Ferme la socket du client actuel
```

CODE SERVEUR 3

serveur.c × client.c × Makefile

```
170                     close(client->sock); // Ferme la socket du client actuel
171                     close(other_client->sock); // Ferme la socket du deuxième client
172                     fin_jeu = NB_JOUEURS;
173                     pthread_cond_broadcast(&cond);
174                 } else if (est_plein()) {
175                     char message[LIGNE_MAX];
176                     envoyer_grille(client->sock);
177                     envoyer_grille(other_client->sock);
178                     sprintf(message, "\nMatch Nul\n");
179                     send(data->clients[0].sock, message, strlen(message), 0);
180                     send(data->clients[1].sock, message, strlen(message), 0);
181                     close(client->sock); // Ferme la socket du client actuel
182                     close(other_client->sock); // Ferme la socket du deuxième client
183                     fin_jeu = NB_JOUEURS;
184                     pthread_cond_broadcast(&cond);
185                 } else {
186                     joueur_actuel = (joueur_actuel == 1) ? 2 : 1;
187                 }
188             }
189             pthread_mutex_unlock(&mutex);
190             pthread_cond_broadcast(&cond);
191         }
192     }
193 }
194
195 // Déconnexion des clients après la fin du jeu (commenté pour éviter une double déconnexion)
196 close(client->sock);
197 close(other_client->sock);
198 return NULL;
199 }
200
201
202 void initialiser_grille() {
203     for (int i = 0; i < TAILLE_GRILLE; i++) {
204         for (int j = 0; j < TAILLE_GRILLE; j++) {
205             grille[i][j] = ' ';
206         }
207     }
208 }
209
210 void afficher_grille() {
211     for (int i = 0; i < TAILLE_GRILLE; i++) {
212         for (int j = 0; j < TAILLE_GRILLE; j++) {
213             printf("%c", grille[i][j]);
```

```
210 void afficher_grille() {
211     for (int i = 0; i < TAILLE_GRILLE; i++) {
212         for (int j = 0; j < TAILLE_GRILLE; j++) {
213             printf("%c", grille[i][j]);
214             if (j < TAILLE_GRILLE - 1) {
215                 printf("|");
216             }
217         }
218         printf("\n");
219         if (i < TAILLE_GRILLE - 1) {
220             printf("———\n");
221         }
222     }
223 }
224
225 int verifier_gagnant() {
226     for (int i = 0; i < TAILLE_GRILLE; i++) {
227         if (grille[i][0] == grille[i][1] && grille[i][1] == grille[i][2] && grille[i][0] != ' ') {
228             return 1;
229         }
230         if (grille[0][i] == grille[1][i] && grille[1][i] == grille[2][i] && grille[0][i] != ' ') {
231             return 1;
232         }
233     }
234     if (grille[0][0] == grille[1][1] && grille[1][1] == grille[2][2] && grille[0][0] != ' ') {
235         return 1;
236     }
237     if (grille[0][2] == grille[1][1] && grille[1][1] == grille[2][0] && grille[0][2] != ' ') {
238         return 1;
239     }
240     return 0;
241 }
```

CODE SERVEUR 4

```
242
243 int est_plein() {
244     for (int i = 0; i < TAILLE_GRILLE; i++) {
245         for (int j = 0; j < TAILLE_GRILLE; j++) {
246             if (grille[i][j] == ' ') {
247                 return 0;
248             }
249         }
250     }
251     return 1;
252 }
253
254 void envoyer_grille(int sock) {
255     char buffer[LIGNE_MAX];
256     snprintf(buffer, LIGNE_MAX, "Grille:\n");
257     for (int i = 0; i < TAILLE_GRILLE; i++) {
258         for (int j = 0; j < TAILLE_GRILLE; j++) {
259             snprintf(buffer + strlen(buffer), LIGNE_MAX - strlen(buffer), "%c", grille[i][j]);
260             if (j < TAILLE_GRILLE - 1) {
261                 snprintf(buffer + strlen(buffer), LIGNE_MAX - strlen(buffer), "|");
262             }
263         }
264         snprintf(buffer + strlen(buffer), LIGNE_MAX - strlen(buffer), "\n");
265         if (i < TAILLE_GRILLE - 1) {
266             snprintf(buffer + strlen(buffer), LIGNE_MAX - strlen(buffer), "———\n");
267         }
268     }
269     snprintf(buffer + strlen(buffer), LIGNE_MAX - strlen(buffer), "Tour du joueur %d\n", joueur_actuel);
270     send(sock, buffer, strlen(buffer), 0);
271 }
```

CODE CLIENT

client.c

```c
#include "pse.h"

#define CMD "client"

int main(int argc, char *argv[]) {
    int sock, ret;
    struct sockaddr_in adrServ;
    int fin = 0;
    char ligne[LIGNE_MAX];
    char buffer[LIGNE_MAX];

    if (argc != 3)
        erreur("usage: %s machine port\n", argv[0]);

    printf("%s: creating a socket\n", CMD);
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
        erreur_IO("socket");

    printf("%s: DNS resolving for %s, port %s\n", CMD, argv[1], argv[2]);
    adrServ = *resolv(argv[1], argv[2]);
    if (adrServ.sin_addr.s_addr == 0)
        erreur("adresse %s port %s inconnus\n", argv[1], argv[2]);

    printf("%s: adr %s, port %hu\n", CMD,
           stringIP(ntohl(adrServ.sin_addr.s_addr)),
           ntohs(adrServ.sin_port));

    printf("%s: connecting the socket\n", CMD);
    ret = connect(sock, (struct sockaddr *)&adrServ, sizeof(adrServ));
    if (ret < 0)
        erreur_IO("connect");


    ret = recv(sock, buffer, LIGNE_MAX, 0);
        if (ret < 0) {
            erreur_IO("recv");
        }
    buffer[ret] = '\0';
    printf("\n%s\n", buffer);

```

Makefile

```c
    while (!fin) {
        // Recevoir la grille du serveur
        ret = recv(sock, buffer, LIGNE_MAX, 0);
        if (ret < 0) {
            erreur_IO("recv");
        }
        buffer[ret] = '\0';
        printf("Grille reçue:\n%s\n", buffer);

        // Analyser les données reçues pour déterminer le tour du joueur
        if (strstr(buffer, "Tour du joueur 1")) {
            if (strcmp(CMD, "client1") == 0) {
                printf("C'est le tour du joueur 1.\n");
            }
        } else if (strstr(buffer, "Tour du joueur 2")) {
            if (strcmp(CMD, "client2") == 0) {
                printf("C'est le tour du joueur 2.\n");
            }
        }

        // Demander un mouvement au joueur
        printf("ligne> ");
        if (fgets(ligne, LIGNE_MAX, stdin) == NULL)
            erreur("saisie fin de fichier\n");

        ret = ecrireLigne(sock, ligne);
        if (ret == -1)
            erreur_IO("ecrire ligne");

        printf("%s: %d bytes sent\n", CMD, ret);

        if (strcmp(ligne, "fin\n") == 0)
            fin = 1;
    }


    if (close(sock) == -1)
        erreur_IO("close socket");

    exit(EXIT_SUCCESS);
}
```

MAKEFILE

Makefile

```
1 # TP2 : Fichier Makefile
2 #
3 include ../Makefile.inc
4
5 EXE = projet serveur client
6
7 ${EXE): ${PSE_LIB}
8
9 all: ${EXE}
10
11 clean:
12         rm -f *.o *~ ${EXE}
13
```