# 1) TP – Couche 4 OSI – Client / Serveur UDP (mode non connecté)

Le projet sera de vous faire développer un jeu de morpion « simple » sur une grille 3x3. Les échanges entre les applications des joueurs, se feront via des datagrammes, et le protocole réseau alors utilisé est UDP.

L'objectif de ce TP est précisément de vous faire découvrir et développer ces « outils » communiquant au travers le réseau. Vous programmerez sous Linux en utilisant l'API-socket en C. Tous les programmes que vous développerez reposent une architecture de type client/serveur (C/S) : le client faisant des requêtes, le serveur y répondant. L'accent sera ici mis sur l'aspect programmation réseau (le service envisagé étant très réduit).
Ici, l'échange C/S repose sur l'utilisation du protocole UDP. Ce niveau de service réseau permet le minimum : le client envoie un ou des paquets (on parle de « datagrammes »). Ceux-ci voyagent de façon indépendante et rapide sur le réseau car il n'y a pas de contrôle effectué (erreurs, séquencement, ….).

### Etape 1 : Etude du schéma global

L'architecture des 2 programmes à construire est montrée à la figure 1, et la figure 2 représente les « sockets » obtenus.

Questions :

- Rappelez ce qu'est une prise réseau « socket », et quels sont les 3 éléments qui la caractérisent.
- Expliquez ce que contient le fichier */etc/services*
- A l'aide de la commande *man*, explorez les définitions des fonctions *socket()*, *bind()*, *recvfrom()*, *sendto()*, et *close()*, en répondant aux questions :
  - Quel est leur rôle ? Sont-elles bloquantes ? Quelles sont leurs arguments ? Que faut-il à *socket()* pour préciser les modes de fonctionnement IPv4 et UDP ?
  - Quelle est la différence fondamentale entre le n° de port coté serveur et celui coté client ?
  - A quoi sert une variable de type structuré *struct sockaddr* ?
  - A quoi peuvent servir des fonctions standards comme *htonl(), htons(), inet_addr() ?*
- On considère un service très simple d'écho (c.à.d le serveur renvoie au client son message) :
  - Détaillez la séquence d'échanges entre les 2 entités C/S.
  - Explicitez le mode non connecté (ou datagramme).

### Etape 2 : Programmation du serveur et du client, fonctionnement sur la boucle locale

Dans un 1er temps, vous ferez fonctionner vos C/S sur la boucle locale (cf. réseau *127.0 0.0/8*). Le service doit être un simple service d'écho où le serveur renvoie au client son message (une chaîne de caractères sans espace, au maximum 128 caractères) .

Manipulations 1 :

- Codez le fichier *serveurUDP.c*
- Compilez-le : *$ gcc -Wall -o serveurUDP serveurUDP.c*
- Lancez le : *$ ./serveurUDP*
  et vérifiez que le serveur « écoute » : *$ netstat -ulnp*

Manipulations 2 :

- Codez le fichier *clientUDP.c*
- Compilez-le : *$ gcc -Wall -o clientUDP clientUDP.c*
- Vérifiez l'échange avec le serveur

### Etape 3 : Programmation du serveur et du client, fonctionnement dans le réseau de la salle TP

Manipulations :

- Reprenez les codes de l'étape 2, mais en faisant fonctionner les C/S entre 2 PC différents de la salle TP.

### Etape 4 : Programmation d'un *Makefile*

La commande *make* permet de lancer toute une série de commandes par le système, exactement ce dernier interprète le script écrit par défaut dans le fichier *Makefile* qui est de type texte ascii, *make* fait aussi tout un ensemble de tests (par exemple la vérification des dates de création des sources). C'est très utile notamment pour lancer des compilations, un exemple vous est donné en annexe.

Manipulations :

- Ecrivez le *Makefile* permettant de lancer les compilations de votre client et de votre serveur.
- Finissez votre code en plaçant dans une archive (*zip* ou *tar*) vos sources **commentées**, le *Makefile*. et un fichier texte *ReadMe.txt* expliquant comment lancer vos algorithmes.

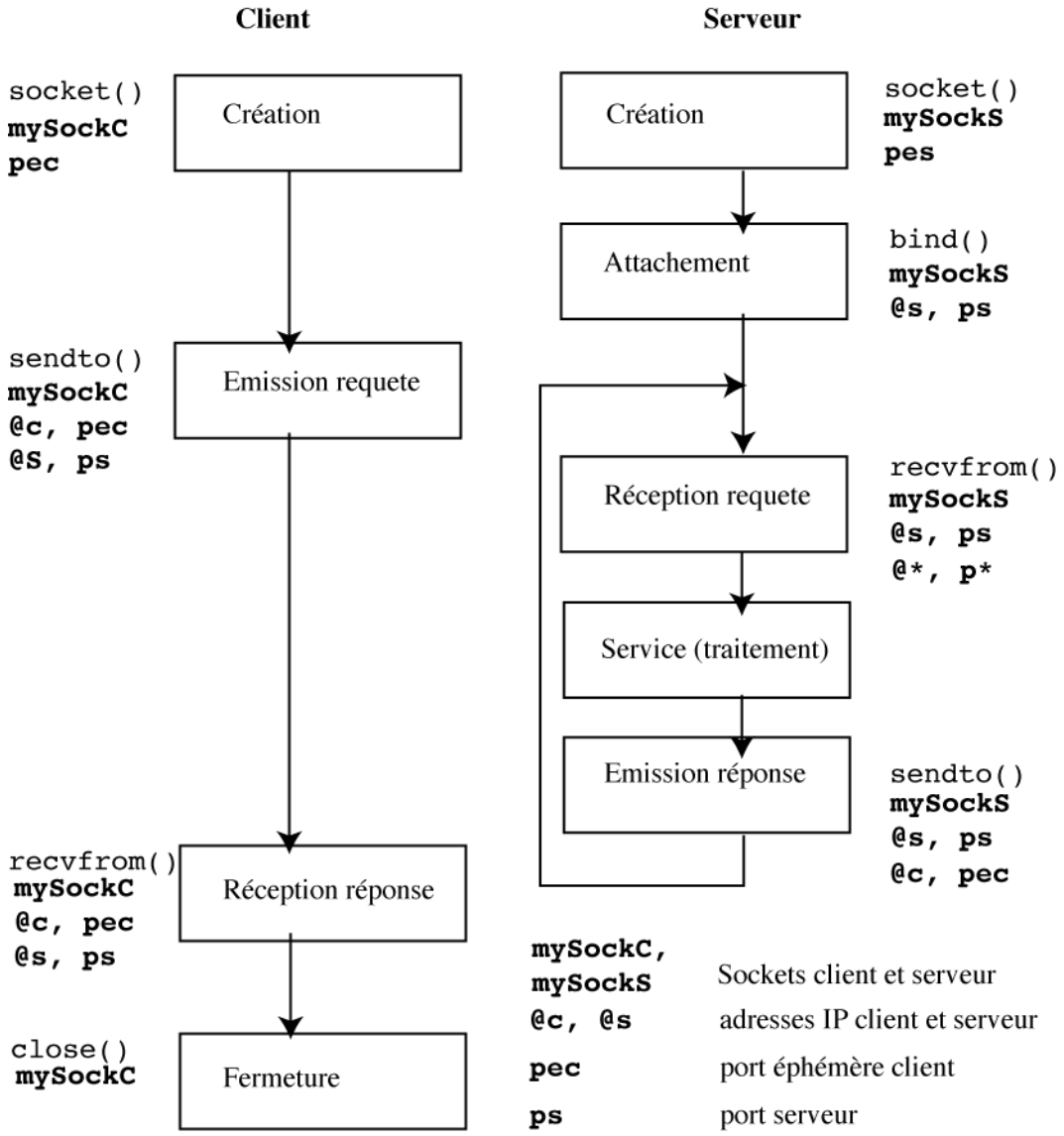


*Figure 1: L'architecture Client/Serveur UDP.*

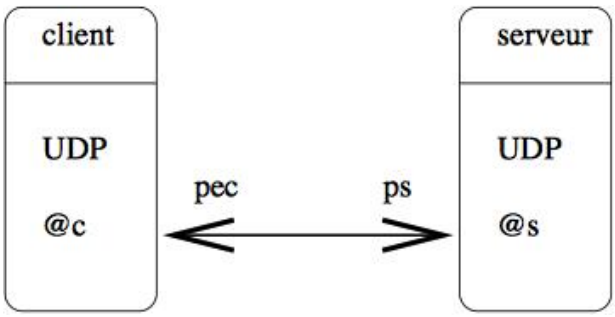


*Figure 2: Client/Serveur UDP.*

## Annexe - *Makefile*

Dans le fichier ascii *Makefile* sont placés des items du type :

***cible**: **dépendances***
    ***commandes** (expliquant comment réaliser la **cible** à partir des **dépendances**)*

Par exemple, si l'exécutable dépend des 2 fichiers source *fich1.c* et *fich2.c*, et fait appel aux fonctions mathématiques de la librairie standard. Le *Makefile* sera :

```
all: executable clean

executable: fich1.o fich2.o
    gcc -Wall -o executable fich1.o fich2.o -lm

fich1.o: fich1.c
    gcc -Wall -c fich1.c

fich2.o: fich2.c
    gcc -Wall -c fich2.c

clean:
    rm *.o *.* ~
```

Ça ce complique car la commande *make* use de règles implicites, de symboles (par exemple : *$@*, *$^* ), et impose une syntaxe rigoureuse. Dans ce cas, le *Makefile* précédent devient :

```
CC=gcc

all: executable clean

executable: fich1.o fich2.o
    $(CC) -Wall -o $@ $^ -lm

clean:
     rm *.o *.* ~
```

La commande pour lancer l'interprétation du script est :

- comportement par défaut avec recherche du fichier *Makefile* dans le répertoire courant :
  *$ make*
- comportement avec recherche du fichier *makefile-cible* au bout de */chemin* :
  *$ make -f /chemin/makefile-cible*

Pour en savoir plus regardez : *$ info make* ou *$ man make*

## 2) Le jeu du Morpion

La grille (3x3) du jeu est présentée à la figure 3 qui indique aussi le système de coordonnées utilisé pour repérer un élément jouer.
Vous pouvez facilement afficher une telle grille dans le terminal sans utiliser d'interface.

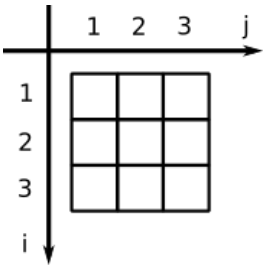

*Figure 3: La grille (3x3) du morpion.*

Quelques conventions :

- Le serveur UDP est sur la boucle locale (*lo*) et écoute sur le port 8003 ;
- Le joueur « client » joue le 1er, puis le joueur « serveur », puis le « client », etc ... jusqu'à la fin de la partie (car « gagnée » ou « à égalité ») ;
- La vérification « Partie gagnée » (3 éléments alignés dans la grille) ou « Egalité » (grille pleine) est faite visuellement par le joueur après avoir joué ;
- Les applications échangent à tour de rôle les coordonnées de l'élément joué, soient 2 entiers précisant l'indice des lignes *(i = 1, 2, ou 3)* et l'indice des colonnes *(j = 1, 2, ou 3)*. Exactement l'application envoie d'abord l'entier précisant *i*, puis l'entier suivant précisant *j*.
- Le joueur pour indiquer la fin d'une partie envoie :
  - *i = 0* puis *j = 0* car il a gagné la partie,
  - *i = 4* puis *j = 4* car il y a « égalité ».
- La figure 4 décrit un diagramme de séquence possible.

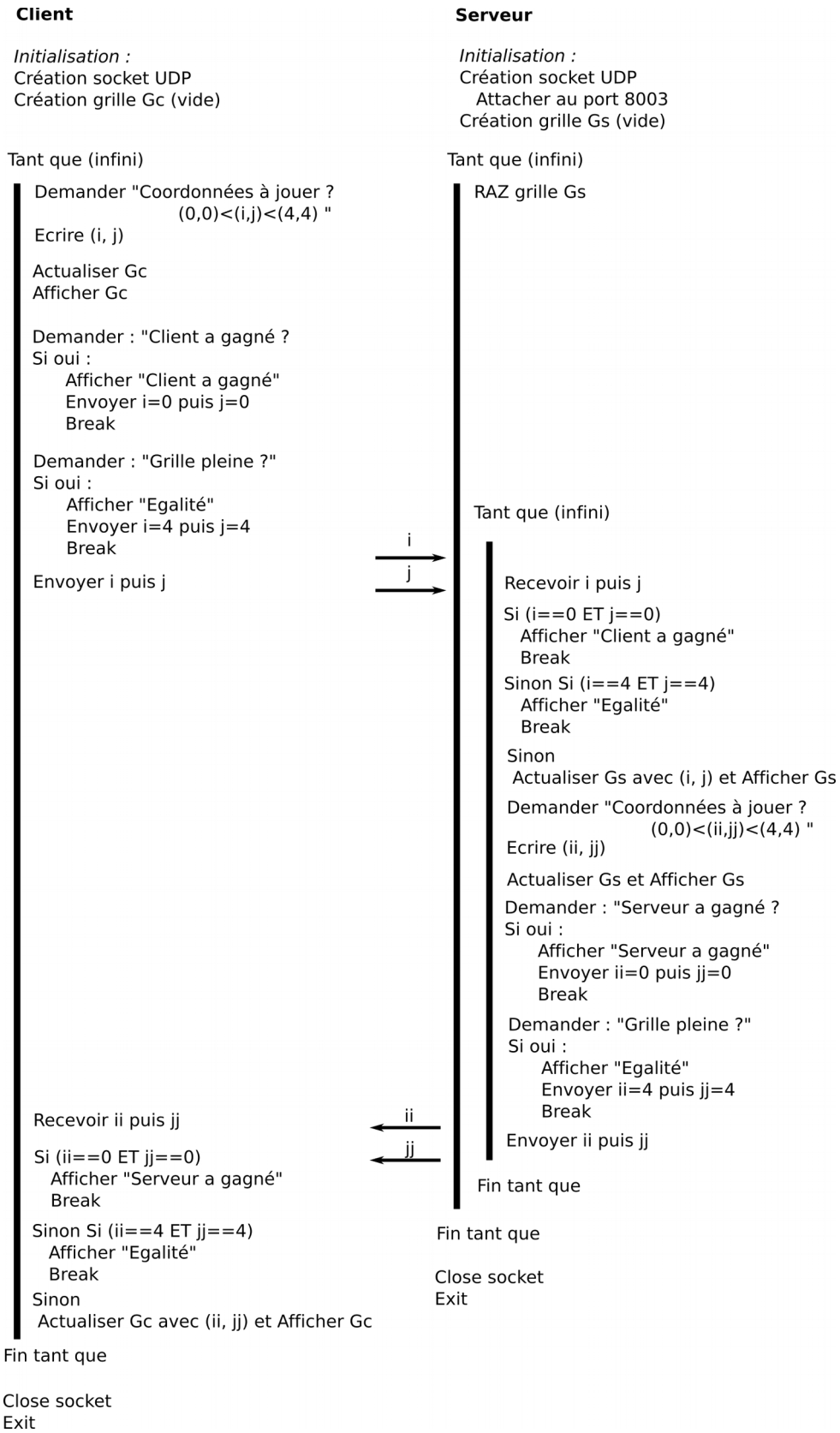


*Figure 4: Diagramme de séquence du jeu du morpion.*