

Université / U.F.R. des Sciences et Techniques

Compte-Rendu de TP Réseau

TP5 : Programmation de Sockets UDP (C et Java)

Auteur :

REDJRADJ YACINE

Année Universitaire 2025-2026

Table des matières

1	Rappel du sujet	2
2	Analyse du sujet	2
3	Choix techniques effectués	2
4	Résultats et tests	2
4.1	Interopérabilité C / Java	2
4.2	Jeu de Morpion	3
5	Conclusion	3

1 Rappel du sujet

L'objectif de ce TP est de maîtriser la programmation réseau via les sockets en mode non-connecté (UDP). Les tâches principales réalisées sont :

- Implémentation d'un serveur et d'un client en langage C échangeant un message texte et un identifiant numérique.
- Portage de l'application en JAVA pour tester l'interopérabilité entre les deux environnements.
- Développement d'un jeu de Morpion (Tic-Tac-Toe) en réseau où le serveur Java gère la logique de jeu et une IA aléatoire.

2 Analyse du sujet

La communication par sockets UDP (*User Datagram Protocol*) se distingue par son absence de connexion persistante. Chaque envoi est un paquet autonome, ce qui impose une rigueur particulière dans le formatage des données applicatives.

L'un des points critiques analysés lors de ce TP est la gestion de l'**Endianness**. Le langage C sur architecture x86 traite les données en *Little-Endian*, tandis que Java tend vers le *Big-Endian* par défaut pour le réseau. Pour que le client Java puisse interpréter correctement les entiers envoyés par le serveur C, il a fallu configurer explicitement l'ordre des octets lors de la lecture du buffer.

3 Choix techniques effectués

L'architecture du projet suit une organisation modulaire :

- **C POSIX** : Utilisation de la structure `sockaddr_in` et de la fonction `recvfrom()` pour récupérer l'adresse de l'expéditeur et lui répondre.
- **Java Datagram** : Utilisation de `DatagramSocket`. Pour le jeu, le choix s'est porté sur l'envoi de chaînes de caractères (**String**) délimitées par un séparateur "|", permettant de transmettre simultanément l'état de la grille et le statut de la partie.
- **Logique de jeu** : Le serveur centralise la vérification des conditions de victoire (lignes, colonnes et diagonales) et assure la cohérence du plateau avant de notifier le client.

4 Résultats et tests

4.1 Interopérabilité C / Java

Les tests ont validé la capacité d'un client Java à interroger un serveur C. La conversion des types de données a été un succès, permettant l'affichage du numéro de client côté Java après réception d'un datagramme binaire issu du C.

```

yacine@yacinelaptop: ~/Bureau/RESEAU/TP5-RedjradjYacine
yacine@yacinelaptop:~/Bureau/RESEAU/TP5-RedjradjYacine$ ./target/bin/helloServer
Lancement du serveur sur le port 5000
Bonjour du client Java
Le serveur envoie le numéro de client 1
[]

yacine@yacinelaptop:~/Bureau/RESEAU/TP5-RedjradjYacine$ javac -d target/
classes src/main/java/HelloClient.java
yacine@yacinelaptop:~/Bureau/RESEAU/TP5-RedjradjYacine$ java -cp target/
classes HelloClient localhost 5000
Envoi du message au serveur...
Attente de la réponse du serveur...
Le numéro attribué par le serveur C est : 1
yacine@yacinelaptop:~/Bureau/RESEAU/TP5-RedjradjYacine$ []

```

FIGURE 1 – Capture d’écran de l’échange réussi entre le Serveur C et le Client Java

4.2 Jeu de Morpion

La figure 2 illustre une session complète du jeu de Morpion. On observe l’initialisation du jeu, la saisie des index par l’utilisateur, et la réponse automatique du serveur plaçant ses jetons (O). La partie se termine par la détection correcte de la victoire du joueur (WIN_PLAYER).

```

yacine@yacinelaptop: ~/Bureau/RESEAU/TP5-RedjradjYacine$ java -cp target/classes morpion.MorpionClient
--- JEU DU MORPION ---
Entrez un index (0-8) : 4

. | . | .
-----
0 | X | .
-----
. | . | .

Entrez un index (0-8) : 1

. | X | O
-----
0 | X | .
-----
. | . | .

Entrez un index (0-8) : 7

. | X | O
-----
0 | X | .
-----
. | X | .

FIN : WIN_PLAYER
yacine@yacinelaptop:~/Bureau/RESEAU/TP5-RedjradjYacine$ []

```

FIGURE 2 – Déroulement d’une partie : affichage de la grille et détection de victoire

5 Conclusion

Ce TP a permis d’appréhender concrètement les problématiques de bas niveau liées aux réseaux, notamment la sérialisation des données et la compatibilité entre différents langages de programmation. L’utilisation de l’UDP s’est révélée efficace pour un jeu simple, malgré l’absence de contrôle de flux natif.

Difficultés rencontrées : L’ajustement du type `socklen_t` dans les fonctions C et

la manipulation des buffers `ByteBuffer` en Java pour respecter l'ordre des octets ont été les points les plus délicats.

Perspectives : Une extension intéressante consisterait à implémenter un serveur capable de gérer plusieurs parties simultanées en utilisant des identifiants de session uniques pour chaque couple client/serveur.