**INGINIERIE INFORMATIQUE ET SYSTEMES EMBARQUES**

**COMPTE RENDU**

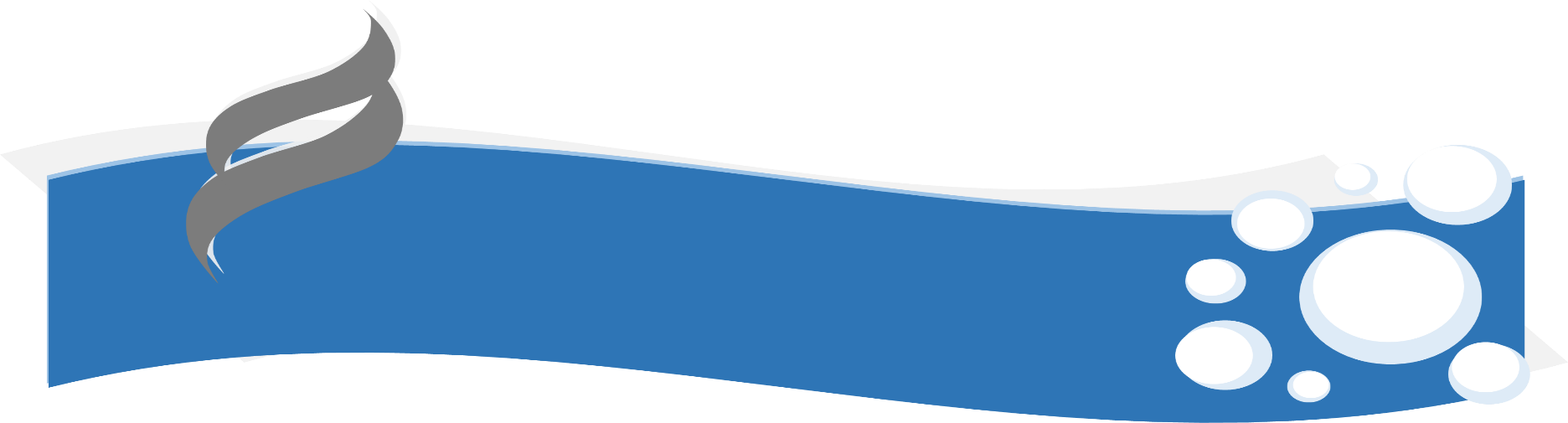
Effectué du : 25/04/2025

tP4 : Systems repartis et distribues

**Réalisé par :**

**-***SARMA FATIMA ZAHRA*

|  |
| --- |
| **Encadrant  :** |
| * *ZAKARIA RIDA* |



**Année Universitaire**

2024/2025

***Contents***

[I. Introduction : 4](#_Toc196496413)

[II. Exercice 4 : 4](#_Toc196496414)

[A. Description du Travail Réalisé 4](#_Toc196496415)

[B. Résultats et Analyse: 6](#_Toc196496416)

[III. conclusion 8](#_Toc196496417)

***Tables des figures***

[Figure 1: le programme de serveur RPC 5](#_Toc196496675)

[Figure 2: le programme des clients 6](#_Toc196496676)

[Figure 3: Demarrage du systeme RPC 6](#_Toc196496677)

[Figure 4: Execution de programme serveur 6](#_Toc196496678)

[Figure 5: devinez un nombre 7](#_Toc196496679)

[Figure 6: Deconnection des clients 7](#_Toc196496680)

# Introduction :

Dans ce rapport, nous allons faire quelque chose d’amusant, mais cette fois en utilisant la **(RPC)** grâce à la bibliothèque **Pyro4**. Le jeu est un petit jeu de devinettes excitant où cinq clients se connectent à un serveur RPC, chacun essayant de deviner un nombre entre **1 et 100** pour correspondre au nombre aléatoire choisi par le serveur.

Le serveur attribue à chaque client un nombre cible unique dès sa connexion et attend les propositions des clients. Le jeu suit le déroulement suivant :

* Si le nombre proposé est inférieur au bon nombre, la réponse est : "Too Low!"
* Si le nombre proposé est supérieur au bon nombre, la réponse est : "Too High!"
* Si le nombre proposé est correct, la réponse est : "Ding Ding Ding! Correct, You have WON!"

# Exercice 4 :

## Description du Travail Réalisé

Dans cette section, nous approfondirons la manière de créer chaque programme, comment connecter les clients au serveur RPC, et verrons quel devrait être le résultat attendu.

1. Le programme de serveur RPC :

Note!: Dans l’autre TP, nous devions inclure la bibliothèque **threading** pour accepter plusieurs clients et travailler avec eux simultanément. Mais avec **RPC**, tout devient plus simple, car il n’est plus nécessaire d’utiliser **threading** : la gestion des connexions multiples est déjà intégrée et prise en charge automatiquement.

import Pyro4

import random

@Pyro4.expose

class GameServer:

    def \_\_init\_\_(self):

        self.clients = {}  # Store each client's target number

    def handle\_client(self, client\_id):

        """Register a new client and generate a random number."""

        target\_number = random.randint(1, 100)

        self.clients[client\_id] = target\_number

        return "Welcome! Guess a number between 1 and 100"

    def start\_game(self, client\_id, guess):

        """Check the client's guess."""

        if client\_id not in self.clients:

            return "Client not registered. Please register first."

        try:

            guess = int(guess)

        except ValueError:

            return "Please enter a valid number."

        target\_number = self.clients[client\_id]

        if guess < target\_number:

            return "Too Low!"

        elif guess > target\_number:

            return "Too High!"

        else:

            del self.clients[client\_id]

            return "Ding Ding Ding! Correct, You have WON!"

def start\_server():

    daemon = Pyro4.Daemon()

    ns = Pyro4.locateNS()

    uri = daemon.register(GameServer())

    ns.register("GameServer", uri)

    print("Server is running...")

    daemon.requestLoop()

if \_\_name\_\_ == "\_\_main\_\_":

    start\_server()

    threading.Thread(target=handle\_client, args=(client\_socket, client\_address)).start()

Figure 1: le programme de serveur RPC

1. Le programme des clients :

Tous les programmes des clients sont le même :

import Pyro4

# Locate the name server

ns = Pyro4.locateNS()

uri = ns.lookup("GameServer")

# Create proxy to the remote GameServer

server = Pyro4.Proxy(uri)

# Interact with the server

client\_id = "client1"

print(server.handle\_client(client\_id))  # Register client

# Play the game

while True:

    guess = input("Enter your guess: ")

    result = server.start\_game(client\_id, guess)

    print(result)

    if "Correct" in result:

        break

Figure 2: le programme des clients

## Résultats et Analyse:

Dans un premier terminal, faire le demarrage du serveur de nom Pyro4

Executer la commande suivante pour le demarrage:

pyro4-ns

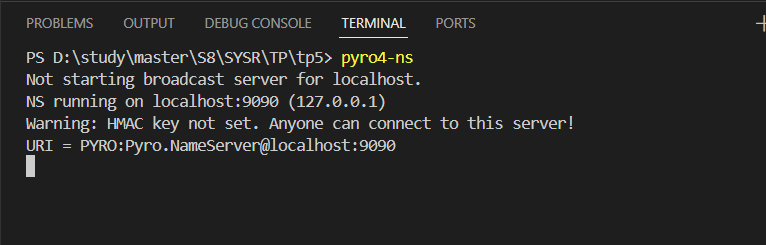


Figure 3: Demarrage du systeme RPC

Dans un deuxieme terminal, executer le programme de serveur par la commande :

python server.py

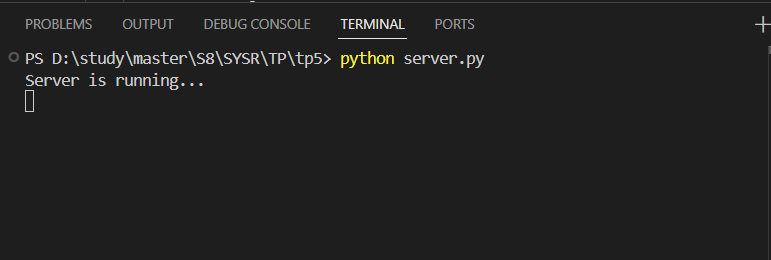


Figure 4: Execution de programme serveur

Dans un troisieme terminal, executer le programme de client par la commande:

python client.py

****









Figure 5: devinez un nombre

Une fois que le client devine le bon numéro après plusieurs échecs, il se déconnectera automatiquement du serveur, Le même chose pour les autres programmes de clients.

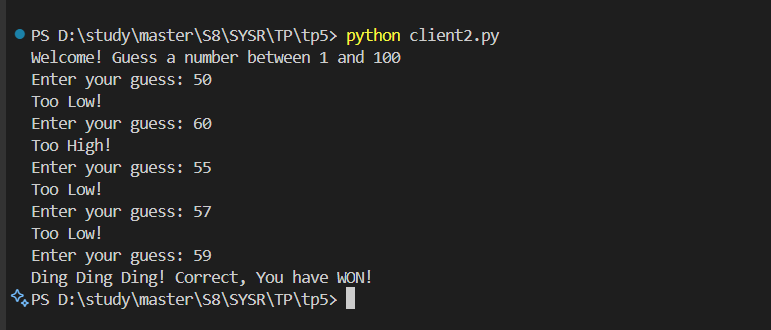


Figure 6: Deconnection des clients

# conclusion

Le passage des sockets à RPC avec Pyro4 a grandement simplifié le projet et accéléré son développement. Au lieu de gérer manuellement les connexions, les threads et l’échange de messages, nous avons pu nous concentrer sur la logique du jeu. Grâce à Pyro4, les appels à distance fonctionnent comme des appels de fonctions locales, rendant le code plus clair, plus simple et plus maintenable, tout en facilitant la gestion de plusieurs clients.