Compte Rendu : TP3 Programmation Événementielle

Module R309

Programmation événementielle

Taha Adam

Exercice 1 : ClientServeur

Objectifs

- Implémenter une classe abstraite `ClientServeur` avec des méthodes pour envoyer et recevoir des messages.
- Développer une classe `ClientServeurUDP` basée sur UDP pour envoyer et recevoir des messages.
- Créer un serveur d'écho, `EchoUDPServer`, qui renvoie aux clients leur message préfixé par l'adresse et le port du serveur.
- Proposer un client, `EchoUDPClient`, capable d'envoyer des messages à un serveur d'écho et d'afficher les réponses.
- Implémenter un serveur multi-thread UDP `ServeurUDPMT` et une version multi-thread du serveur d'écho, `EchoUDPServerMT`.

Code Complet

```python
from abc import ABC, abstractmethod
import socket
import threading

# Classe abstraite ClientServeur
class ClientServeur(ABC):

    @abstractmethod
    def envoyerMsg(self, message, adresse):
        pass

    @abstractmethod
    def recevoirMsg(self):
        pass

# Classe abstraite ClientServeurUDP héritant de ClientServeur
class ClientServeurUDP(ClientServeur):
    def __init__(self, host='', port=9000):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((self.host, self.port))

```

```python
    def envoyerMsg(self, message, adresse):
        self.sock.sendto(message.encode(), adresse)

    def recevoirMsg(self):
        data, addr = self.sock.recvfrom(1024)
        return data.decode(), addr

# Classe EchoUDPServer héritant de ClientServeurUDP
class EchoUDPServer(ClientServeurUDP):
    def start(self):
        print(f"Serveur Echo UDP démarré sur {self.host}:{self.port}")
        while True:
            message, client_addr = self.recevoirMsg()
            response = f"Echo de {self.host} {self.port}: {message}"
            self.envoyerMsg(response, client_addr)

# Classe EchoUDPClient héritant de ClientServeurUDP
class EchoUDPClient(ClientServeurUDP):
    def envoyerMsg(self, message, adresse):
        super().envoyerMsg(message, adresse)
        response, _ = self.recevoirMsg()
        print(f"Réponse du serveur: {response}")

# Classe abstraite ServeurUDPMT (Multi-Thread)
class ServeurUDPMT(ClientServeurUDP, ABC):
    def __init__(self, host='', port=9001):
        super().__init__(host, port)

    @abstractmethod
    def traiterClient(self, message, client_addr):
        pass

    def start(self):
        print(f"Serveur UDP Multi-Thread démarré sur {self.host}:
{self.port}")
        while True:
            message, client_addr = self.recevoirMsg()
            thread = threading.Thread(target=self.traiterClient, args=
(message, client_addr))
            thread.start()

# Classe EchoUDPServerMT héritant de ServeurUDPMT
class EchoUDPServerMT(ServeurUDPMT):
    def traiterClient(self, message, client_addr):
        thread_name = threading.current_thread().name
        response = f"[{thread_name}] Echo: {message}"
        self.envoyerMsg(response, client_addr)

# Exemple d'utilisation
if __name__ == "__main__":
    choix = input("Mode Serveur (S), ou Client (C) ? : ").lower()
if choix == 's':
    mode = input("Mode simple (1) ou multi-thread (2) ? : ")
    if mode == '1':
        serveur = EchoUDPServer(host='', port=9000)  # Serveur simple sur le
port 9000
```

```
        serveur.start()
    elif mode == '2':
        serveur_mt = EchoUDPServerMT(host='', port=9001)  # Serveur multithread sur le port 9001
        serveur_mt.start()
elif choix == 'c':
    client = EchoUDPClient(host='', port=0)  # Port client dynamique
    serveur_host = input("Entrez l'adresse IP du serveur : ")
    serveur_port = int(input("Entrez le port du serveur : "))
    while True:
        message = input("Message à envoyer : ")
        client.envoyerMsg(message, (serveur_host, serveur_port))

```

Partie 1: Classe abstraite `ClientServeur`

```python
class ClientServeur(ABC):
    @abstractmethod
    def envoyerMsg(self, message, adresse):
        pass

    @abstractmethod
    def recevoirMsg(self):
        pass
```

- Cette classe définit une **interface commune** pour toutes les implémentations de client ou de serveur.

  Les deux méthodes abstraites `envoyerMsg` et `recevoirMsg` établissent une structure

---

Partie 2 : Classe `ClientServeurUDP`

```python
class ClientServeurUDP(ClientServeur):
    def __init__(self, host='', port=9000):
        """Initialise un socket UDP lié à une adresse et un port."""
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind((self.host, self.port))
```

- Hérite de la classe `ClientServeur`.
- Implémente un socket UDP qui permet d'envoyer et de recevoir des messages.

Méthodes `envoyerMsg`

```
    def envoyerMsg(self, message, adresse):
        self.sock.sendto(message.encode(), adresse)

```

- Permet d'envoyer un message encodé

Méthodes `recevoirMsg`

```
    def recevoirMsg(self):
        data, addr = self.sock.recvfrom(1024)
        return data.decode(), addr
```

- Reçoit un message d'une taille maximale de 1024 octets et retourne les données

---

Partie 3 : Classe `EchoUDPServer`

```
class EchoUDPServer(ClientServeurUDP):
    def start(self):
        print(f"Serveur Echo UDP démarré sur {self.host}:{self.port}")
        while True:
            message, client_addr = self.recevoirMsg()
            response = f"Echo de {self.host} {self.port}: {message}"
            self.envoyerMsg(response, client_addr)
```

- Hérite de `ClientServeurUDP`.
- Implémente un **serveur Echo simple** qui retourne les messages reçus

---

Partie 4 : Classe `EchoUDPClient`

```
class EchoUDPClient(ClientServeurUDP):
    def envoyerMsg(self, message, adresse):
        super().envoyerMsg(message, adresse)
        response, _ = self.recevoirMsg()
        print(f"Réponse du serveur: {response}")
```

- Implémente un client qui peut :
    - D'envoyer un message texte à un serveur Echo.
    - De recevoir et afficher la réponse du serveur.

Partie 5 : Classe abstraite `ServeurUDPMT`

```python
class ServeurUDPMT(ClientServeurUDP, ABC):
    def __init__(self, host='', port=9001):
        super().__init__(host, port)

    @abstractmethod
    def traiterClient(self, message, client_addr):
        pass
```

- Définit une architecture pour un **serveur multithread UDP**.
- La méthode abstraite `traiterClient` est appelée pour traiter chaque client dans un thread distinct.

Démarrage du serveur

```python
    def start(self):
        print(f"Serveur UDP Multi-Thread démarré sur {self.host}:{self.port}")
        while True:
            message, client_addr = self.recevoirMsg()
            thread = threading.Thread(target=self.traiterClient, args=(message, client_addr))
            thread.start()
```

- Chaque message reçu lance un ***Thread*** dédié au traitement du client

---

Partie 6 : Classe `EchoUDPServerMT`

```python
class EchoUDPServerMT(ServeurUDPMT):
    def traiterClient(self, message, client_addr):
        thread_name = threading.current_thread().name
        response = f"[{thread_name}] Echo: {message}"
        self.envoyerMsg(response, client_addr)
```

- Implémente un **serveur Echo multithread** :
    - Chaque client est traité par un thread différent.
    - Le nom du thread est dans la réponse envoyée au client.

Partie 7 : Tests

Mode Serveur

```
if choix == 's':
    mode = input("Mode simple (1) ou multi-thread (2) ? : ")
    if mode == '1':
        serveur = EchoUDPServer(host='', port=9000)  # Serveur simple sur le port 9000
        serveur.start()
    elif mode == '2':
        serveur_mt = EchoUDPServerMT(host='', port=9001)  # Serveur multithread sur le port 9001
        serveur_mt.start()
```

- Permet de choisir entre un serveur Echo simple ou multithread.
- Lance le serveur selectionné

Mode Client

```
elif choix == 'c':
    client = EchoUDPClient(host='', port=0)  # Port client dynamique
    serveur_host = input("Entrez l'adresse IP du serveur : ")
    serveur_port = int(input("Entrez le port du serveur : "))
    while True:
        message = input("Message à envoyer : ")
        client.envoyerMsg(message, (serveur_host, serveur_port))
```

- Envoi des messages et affiche les réponses reçues