Raspberry : un websocket pour recevoir et envoyer des messages en même temps

Le premier programme de Websocket serveur en python :

- **écoutait** tout d'abord le message envoyé par le client (la page html) qui se connecte,
- avant **d'envoyer** le message de bienvenu (Bonjour).

Pour, **à la fois recevoir et envoyer des messages** sur la **même connexion**, on combine les deux fonctions **send()** et **recv()** en les faisant s'exécuter dans des **tâches parallèles** :

```
#!/usr/bin/env python3
import asyncio
import websockets

# Gestion de l'envoi des messages du serveur au client
async def gestion_envoi_message(websocket):
    i = 0
    # boucle infinie pour envoyer toutes les secondes un message ;
    while True:
       await websocket.send("Bonjour {}".format(nb))
       i = i + 1
       await asyncio.sleep(1)

# Gestion des messages recus du client
async def gestion_reception_message(websocket):
    while True:
        # reception du message d'un client
        message = await websocket.recv()
        print(message)

# fonction lancee à chaque connexion d'un client
async def echange(websocket,path): # définir la fonction comme asynchrone
    #envoyer des messages en parallèle
    envoyer = asyncio.ensure_future(gestion_envoi_message(websocket))
    #recevoir message en parallèle
    recevoir = asyncio.ensure_future(gestion_reception_message(websocket))
    termine, attente = await asyncio.wait(
            [envoyer, recevoir],
            return_when = asyncio.FIRST_COMPLETED,
    )

# Definir la fonction qui sera appelee par le serveur a la connexion d'un client
lancement_serveur = websockets.serve(echange,'10.3.141.1', 5678)
# Creation de la boucle d'evenement (event loop)
loop = asyncio.get_event_loop()
loop.run_until_complete(lancement_serveur)
loop.run_forever()
loop.close()
```

- pour lancer le serveur ouvrez le terminal et lancez l'exéuction du programme python : <code shell> $ python3 serveur.py </code>

Programme javascript du client

Dans la page Web HTML :

- la balise **%%%% affiche les messages envoyés toutes les secondes par le serveur, * un bouton HTML est ajouté pour déclencher à la demande l'envoi d'un message au serveur. <code html> <!DOCTYPE html> <html> <head> <meta charset="UTF-8"> </head> <body> En attente d'un message du serveur
 <input type="button" name="envoi" id="envoi" value="Envoyer message"/> <script>** ***selectionner la balise avec son id var affichemessage = document.getElementById('messagerecu');*** **creation du websocket client vers le websocket serveur du Raspnerry var ws = new WebSocket("ws:*****10.3.141.1:5678/");*** ***ws.onopen = function (event) { ws.send("J'envoie un premier message au serveur."); }; ws.onmessage = function (event) {*** **affiche le message recu dans la balise affichemessage.innerHTML = event.data;**
}; ***Ajout d'un gestionnaire d'événement sur le bouton var envoi = document.getElementById('envoi') envoi.addEventListener('click',envoyerMessage); var numeroMessage = 1;***

fonction qui envoie un message au serveur function envoyerMessage() { ws.send(“J'envoie message ” + numeroMessage + “ au serveur.”); numeroMessage += 1; } </script> </body> </html> </code>

Vous pouvez **à la fois** :

- envoyer un message au serveur
- et **recevoir** en même temps un message du serveur

Pour **visualiser l'envoi et la réception simultanés** des messages, cliquez sur le bouton de la page et observez l'**affichage du print** dans le terminal du Raspberry

===== Envoyer plusieurs message en même temps ===== Il peut être utile d'**envoyer en même temps plusieurs messages** et non les uns à la suite des autres. C'est le cas par exemple pour : * faire **avancer** la voiture et, * faire **tourner** en même temps les roues. Dans la fonction du **programme serveur** qui traite de la **réception** des message vous pouvez lancer des **tâches parallèles** de la manière suivante : <code python> async def gestion*reception*message(websocket): while True: # reception du messsage d'un client message = await websocket.recv() print(message)
lancement de la tache qui gere le deplacement avant/arriere de la voiture
asyncio.get*event*loop().create*task(deplacement(message)) # lancement de la tache qui gere la direction droite ou gauche
asyncio.get*event*loop().create*task(direction(message)) </code> Il reste à écrire les fonctions correspondantes **deplacement()** et **direction()** qui doivent **analyser le contenu du message** pour savoir **le sens du déplacement** (avant ou arrière) ainsi que **la direction à prendre** (droite ou gauche. ==== Les activités ... ====

Je reviens à la liste des activités.