Programmation client-serveur
Python, TCP/IP, services,...

Manuel Munier

Version 4 octobre 2022 (10:30)

Objectifs

Dans cette SAE vous allez développer une application client-serveur aux différentes couches du modèle OSI : UDP, TCP/IP, présentation,... L'objectif est de bien comprendre ce que l'on peut faire et comment on peut le faire au fur et à mesure de l'enrichissement de la strate communication.

Consignes du projet

- Le développement se fera en langage Python. Votre code devra impérativement être commenté et être lisible.
- Vous devrez rédiger un court rapport fournissant la description de votre travail à chaque étape (à chaque couche). Il ne s'agit pas de simplement fournir le code commenté ! À vous d'expliquer la logique de vos programmes et ce qu'apporte telle couche par rapport à la précédente. Bref, il vous faudra prendre un peu de recul...
- Une courte présentation ainsi qu'une démo seront exigées en fin de projet.

NB Comme pour toutes les SAE, il s'agit d'un travail individuel. Pendant les séances vous pouvez bien évidemment vous entraidez. Mais l'évaluation finalr sera faite individuellement.

Travail à réaliser

À vous de choisir quelle application client-serveur vous souhaitez développer. Mais faites en sorte que le cas d'étude que vous aurez retenu soit suffisamment "riche" pour exprimer toutes vos compétences...

1. envoi/réception d'un byte array en UDP
2. envoi/réception d'un byte array en TCP
3. toujours en TCP, envoi/réception de données structurées "simples" (ex : une adresse IP) en TCP $\rightsquigarrow$ conversion d'une syntaxe abstraite (tableau de 4 entiers) en syntaxe concrète (byte array)
4. toujours en TCP, envoi/réception de données structurées au format JSON via une chaîne de caractères
5. toujours en TCP, envoi/réception d'objets (pouvant être liés à d'autres objets) via des mécanismes dit de sérialisation $\rightsquigarrow$ à vous de vous documenter sur le sujet
6. notion de service un peu plus "intelligent"
   (a) multithreading côté serveur $\rightsquigarrow$ un même serveur peut servir plusieurs clients en parallèle ; pour cela, il démarre un thread léger à chaque requête client ; ce thread s'occupe de traiter la requête, ce qui permet au serveur de se remettre immédiatement à l'écoute de nouvelles requêtes entrantes

(b) notion de servie $\rightsquigarrow$ le message envoyé par le client ressemble à un appel de méthode : il indique d'abord quel service il veut interroger, puis fournit les paramètres nécessaires à ce service $\rightsquigarrow$ un même serveur peut ainsi proposer plusieurs fonctionnalités (ex : méthodes GET, POST, HEAD,... d'un serveur web)

(c) authentification des utilisateurs $\rightsquigarrow$ restriction d'accès, notion de session,... $\rightsquigarrow$ un peu comme les sessions en PHP que vous aviez vu l'an passé en R209

Document rédigé en LaTeX sous Linux.