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

Manuel Munier

Version 26 septembre 2023 (08:58)

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 finale sera faite individuellement.

Préambule

Afin de vous entraîner sur la programmation des sockets en Python vous pouvez commencer par développer des exemples d'applications client/serveur comme indiqué ci-dessous :

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 dits de sérialisation $\rightsquigarrow$ à vous de vous documenter sur le sujet
6. 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

Travail à réaliser

L'objectif de cette SAE est de programmer un service un peu plus "intelligent". L'idée est que 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).

Le cas d'étude que nous allons traiter est le suivant :

1. le serveur va gérer une liste de promotions avec, pour chacune d'entre elles, la liste des étudiants inscrits ; chaque étudiant disposera d'une liste de notes (avec coefficients) ; côté serveur, à vous de choisir si vous souhaitez utiliser de simples tableaux, des objets ou une base de données...
2. le client se connectera au serveur via des sockets en TCP pour utiliser les différents services :
   (a) créer une nouvelle promotion
   (b) ajouter un nouvel étudiant dans une promotion
   (c) ajouter une note (avec son coefficient) à un étudiant dans une promotion
   (d) demander le calcul de la moyenne d'un étudiant dans une promotion
   (e) demander le calcul de la moyenne d'une promotion
3. à vous de définir et de programmer l'envoi du message par le client au serveur pour demander l'exécution de tel ou tel service en lui passant les paramètres requis
4. idem pour la réponse du serveur au client
5. le serveur sera bien évidemment multithreadé pour pouvoir traiter les requêtes des clients en parallèle
6. fonctionnalités bonus :
   (a) le client demande au serveur de lui retourner la liste des étudiants d'une promotion spécifique (avec leurs notes)
   (b) authentification des utilisateurs $\rightsquigarrow$ restriction d'accès, notion de session,... $\rightsquigarrow$ un peu comme les sessions en PHP que vous aviez vues l'an passé en R209

Document rédigé en LaTeX sous Linux.