

Travail de session – Les sockets

Modalité : Travail individuel ou en équipe de 2 (les mêmes équipes que pour le TP1 sauf exception)

Date de remise : Mardi le 1er avril 2025.

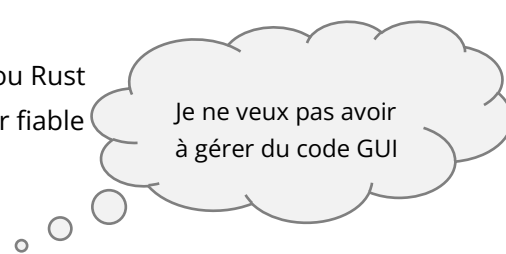
Spécifications techniques de la remise

Pénalité de 20% pour non-respect

- Format du rapport : DOCX ← moins dix pourcent (-10%) si remis en format PDF!
 - Nom du rapport : INF26207_TS_H2025_VotreNom.docx ← -20% sinon!
 - Interligne : 1,15 à 1,3 lignes
 - Police : Times New Roman 11 pt (ou similaire)
 - Utilisez les styles pour séparer vos sections
- Format de la remise du code (vous pouvez aussi utiliser github, gitlab ou autre):
 - Le projet devra être mis dans un fichier zip, 7z, tar.gz ou rar :
INF26207_TS_H2025_VotreNom.extension ← -20% si non-conforme!
 - Utilisez une structure de dossier et des noms de fichiers appropriés pour votre projet/code

Objectifs

- Utilisation des sockets UDP en Python 3.10+ ou Rust
- Implémenter un service de transfert de fichier fiable
- Se familiariser avec la notion de sérialisation



Je ne veux pas avoir
à gérer du code GUI

Énoncé

Vous aurez à implanter un client – **APPLICATION CONSOLE** – (en fait, il pourrait y en avoir plusieurs, mais un seul à la fois) et un serveur UDP.

VOUS NE DEVEZ PAS UTILISER DIRECTEMENT LES CLASSES UDP MAIS PLUTÔT L'IMPLÉMENTATION DE BASE DU SOCKET : telle que, `socket.socket()` dans Python.

- Vous devez implémenter votre solution à l'aide des librairies de base.
- N'utilisez pas plein de librairies externes, à la rigueur, se limiter à TQDM pour les barres de progrès.
- Si vous utilisez d'autres librairies, vous assurer de fournir un fichier requirements.txt pour python et l'équivalent en Rust (cargo)

Fonctionnement général

Vous devez implémenter un serveur de fichier simple de type FTP.

Les commandes reconnues sont les suivantes :

```
open adresse_ip : initie une connexion au serveur spécifié par l'adresse IP
ls : retourne la liste des fichiers disponibles
get nom_de_fichier : retourne le fichier spécifié
bye : termine la connexion au serveur
```

Le serveur écoute sur le port 2212 pour une demande de connexion de la part d'un client.

Lorsqu'un client le contacte, un processus de poignée de main devrait être initié pour confirmer la connexion (pensez au *Three-Way Handshake* du protocole TCP).

Durant le processus de la poignée de main, le client et le serveur devront s'entendre sur :

- La taille maximale de morceaux de fichiers envoyés/reçus.
- Le nombre maximal de morceaux pouvant être envoyés avant envoi d'accusé de réception (fenêtrage, acquittement).

Une fois la connexion établie avec le client, le serveur attend une commande. Si un transfert de fichier est initié, le transfert sera effectué découpant le fichier en morceaux et en utilisant plusieurs commandes SEND. Une fois le fichier transmis, le serveur attend la prochaine commande.

Détails techniques

Le client contacte le serveur afin de recevoir le fichier. Le fichier lui parvient en plusieurs blocs (segments). Le client sera averti d'une façon quelconque de la fin du fichier.

Vous devez donc établir une syntaxe et structure de messages cohérente que le client et le serveur peuvent envoyer/recevoir.

Inspirez-vous des protocoles TCP, HTTP pour la conception et la définition de la structure de vos messages, par exemple :

```

+---+-----+-----+---+-----+-----+
| CMD | <PARAM1> | <PARAM2> | ... | <PARAMN> | <DONNEES> |
+---+-----+-----+---+-----+-----+

```

En somme, vous devez concevoir vos propres en-tête application (encapsulation, PCI [Protocol Control Information] ← module 2).

Les blocs transmis ne doivent pas contenir plus d'octets du fichier que défini lors de l'établissement de la connexion et celui-ci doit faire au moins 200kio. Votre application cliente devra vérifier que le fichier reçu est identique à celui envoyé (fiabilité/*checksum*).

!!! N'utilisez pas l'encodage textuel (ASCII, UTF, ANSI) pour transmettre votre fichier !!!

Votre réseau n'est pas fiable. Afin de simuler cette faiblesse, vous générez aléatoirement des erreurs de transmission. Par exemple, si la fiabilité du réseau (une variable de vos applications qui peut être lue à partir d'un fichier de configuration) est de 0,95, cela signifie que 5 % des messages sont perdus. Pour simuler cela, à l'envoi d'un message, vous rejetterez le message dans 5 % des cas en n'effectuant pas la commande SEND. Évidemment, la fiabilité du réseau est la même pour le serveur et pour le client.

NOTEZ BIEN QUE LA FIABILITÉ DU RÉSEAU DEVRAIT ÊTRE SIMULÉE POUR TOUT ENVOI DE DONNÉES!

La réception de chaque *N* blocs d'information par le client doit être confirmée par un accusé de réception : le client qui reçoit un morceau de fichier doit retourner au serveur un message indiquant qu'il l'a bien reçu. À défaut de recevoir l'accusé de réception d'un bloc de *N* morceaux dans les 3 secondes suivantes (valeur définie dans une constante de votre programme), les morceaux seront réexpédiés par le serveur. Si après 5 tentatives d'envoi consécutives on échoue toujours, on termine le transfert et on avertit le client de l'échec (il pourra, ou pas recevoir ce message en fonction de la fiabilité du réseau...).

Il n'y a pas de réexpédition d'un accusé de réception par le client. Toutefois, si un de ceux-ci se perd, le bloc de *N* morceaux de fichier original sera réexpédié par le serveur, après le délai de 3 secondes, et un nouvel accusé de réception sera donc émis à sa réception.

La réexpédition et la perte de messages peuvent occasionner des doublons. Pour reconnaître le dédoublement des morceaux, il faudrait les numéroter d'une façon quelconque. Lorsqu'on reçoit un doublon, on peut alors le détecter et l'ignorer.

Vous pouvez utiliser une adresse IPv4 de rebouclage (127.0.0.x) pour implémenter et tester vos applications.

Vous aurez aussi à trouver un moyen d'informer le client que tout le fichier a été transmis.

Le client devra connaître le nom du fichier qu'il reçoit.

Livrables

Vous devez remettre un rapport (maximum de 4 pages + annexes) décrivant le fonctionnement de vos applications, vos choix de conception et leurs justifications.

Le rapport devrait inclure la structure et la syntaxe de vos messages

Le travail peut se faire en équipe de deux.

Lors de la correction, vous aurez à faire fonctionner vos applications, répondre à des questions sur leur conception et, peut-être, apporter quelques modifications à votre code.

Barème

- Fonctionnements corrects du client et du serveur → 40 %
 - o Ne plante pas
 - o Résultat attendu
 - o Respecte les requis
- Qualité du rapport → 10 %
 - o Clarté et organisation (5 %)
 - Introduction et conclusion claires (2 %)
 - Les sections sont bien organisées (3 %)
 - o Explication des choix de conception (3 %)
 - Justification des choix techniques (2 %)
 - Explication des compromis (1 %)
 - o Description des difficultés rencontrées et solutions (2 %)
 - Présentation des problèmes rencontrés (1 %)
 - Solutions appliquées pour les résoudre (1 %)
-
- Choix de conception → 25 %
 - o Détails d'implémentation : entêtes, structures, etc.,
 - o Logique,
 - o Etc.
- Qualité de la programmation → 15 %
 - o Commentaires,
 - o Propreté,
 - o Conventions,
 - o Structure des fichiers, des dossiers,
 - o Etc.
- Qualité des interfaces, respect des consignes, etc. 10 %

Notes :

- # Pas besoin de faire des unit test, à moins que ça vous tente, mais je ne les vérifierai pas
- # L'application demandée est simple, un programme trop complexe pourrait être pénalisé
- # Commentez votre code, en particulier si vous définissez des constantes ou autres, je n'aime pas trop jouer aux devinettes. Trop de commentaires sont mieux que pas assez de commentaires dans ce cas.