FilesMirror *version threads*

> **Membres :**
>
> - POTEZ Alexandre
> - RASCLE Jérôme
> - GARCIA Matéo

1. Organisation du projet

Pour ce projet, nous avons utilisé **Github** afin de nous partager le code à chaque fin de sessionde travail. Afin de travailler conjointement, l'extensention **LiveShare** nous a permis de travailler ensemble sur le même fichier. Nous avons également utilisé **Discord** pour communiquer et nous organiser.
Lors de session de travail un des membres du groupe hébergeait un serveur FTP grâce à **FileZilla Server** sur sa machine. Puis, au travers d'un terminal partagé (grâce à **Liveshare** sur **vscode**), tous les membres pouvaient éxecuter le code et en voir le résultat en temps réel.

2. Tâches effectuées

1. Pour se familiariser avec la technologie de **serveurs FTP** (qui nous était dès lors inconnue), nous avons lancé un serveur FTP grâce à **Filezilla Server** puis nous sommes connectés les uns aux autres (via **FileZilla Client**) afins de comprendre le fonctionnement de ce type de serveur.

2. Nous avons ensuite pris connaissance du code existant et de son fonctionnement. En éxecutant le fichier `main.py` suivi des informations de connexion au serveur FTP, nous avons pu constater que le code permettait de se connecter au serveur FTP et de synchroniser les fichiers présents dans un dossier local avec un dossier distant sur le serveur FTP.

3. Une fois le code et sa structure pris en main, nous avons commencé à réfléchir aux méthodes à utiliser pour transformer ce code de manière parallèle. Plusieurs options s'offraient à nous :

   - Le multiprocessing.
   - Le multithreading.
   - La programation asynchrone.

   Notre choix s'est porté sur le **multiThreading** car nous avons trouvé que c'était la méthode la plus simple à mettre en place.

4. Nous avons donc commencé par créer un fichier `applyUpdate.py` qui contient la class `applyUpdate` qui crée un certain nombre de threads (passé en paramètre) et les lancer.

Chaque thread va donc vérifier une Queue nommée `fileToSent`, si cette queue cette se remplie alors le thread va envoyer le fichier à l'aide de la méthode `sendFile`.

5. Nous avons ensuite modifié le fichier `directory_manager.py` afin de créer une Queue nommée `fileToSent` et de l'envoyer en paramètre à `applyUpdate` du fichier `applyUpdate.py`. Nous avons également modifié la méthode `search_updates` pour que celle-ci ajoute les fichiers à la queue `fileToSent` au lieu de les envoyer directement.

6. Pour finir nous avons modifié le nombre d'arguments afin de permettre à l'utilisateur de choisir le nombre de threads à utiliser lors du transferts des fichiers.

3. Difficultés rencontrées

Nous avons rencontré plusieurs difficultés lors de la réalisation de ce projet. Tout d'abord, nous avons eu du mal à comprendre le fonctionnement des serveurs FTP. Nous avons également eu du mal à comprendre le fonctionnement du code existant.

4. Tests effectués

Nous avons testé notre code en créant un dossier `test` dans lequel nous avons créé un fichier `test.txt` et un fichier `test2.txt`. Nous avons ensuite lancé le code avec le nombre de threads à 2. Nous avons constaté que les deux fichiers ont bien été envoyés sur le serveur FTP en parallèle.

5. Comment utiliser le projet

Pour utiliser notre projet, il faut lancer le fichier `main.py` comme pour la version séquentielle. Avec comme paramètre:

- Les informations de connexion au serveur FTP.
- Le fichier local à synchroniser.
- La profondeur de recherche.
- La fréquence de synchronisation.
- **Le nombre de threads à utiliser.**
- Les extensions à exclure (option).

6. Conclusion

Nous avons appris à utiliser le multithreading et à l'appliquer à un projet existant. Nous avons également appris à utiliser les serveurs FTP et à les utiliser dans un projet.