Rapport Projet EL-3032

EL FTOUH Khalid
RAHME Jovan

Sommaire :

I-Explication de nos choix :

Du Côté Client Nous avons décidé de suivre le protocole de communication suivant:
|D|Code Serveur| Code Lieux | Code Menu | NumClient|

Nous avons décidé de suivre ce protocole de communication en ajoutant le NumClient à la fin. Arrivée dans le serveur de routage, la commande se stock dans la table de routage avec comme dernier indice le NumClient, elle est ensuite envoyée au serveur de donné sans le numClient, il sera récupéré au retour de la commande pour déterminer à quel client la renvoyer.

Pourquoi cet entre-deux avec le numéro client ?
Jusqu'à la fin, nous n'utilisions pas de table de routage mais uniquement le code client dans la commande pour se diriger. Finalement, nous avons pu ajouter la table de routage mais pas changer la méthode de fonctionnement avec l'interface client, d'où la présence du numCLient dans la commande venant du client. Néanmoins ,ceci ne cause de problème pour le bon fonctionnement du code

Le serveur de données reçoit la demande du client et commence par la parser afin de récupérer séparément chaque code. Il effectue ensuite une série d'opérations en utilisant ces codes afin d'obtenir le Menu souhaité par le client et lui renvoie à travers le serveur de routage. Un détail: Le serveur de donnée suit le protocole suivant :
R|Code Serveur| Code Lieux | Code Menu | Message|

En cas d'erreur, le serveur de donnée envoie une réponse au serveur de routage sous le format suivant:
E|Code Serveur| Code Lieux | Code Menu | Message d'erreur|
Pour détecter les erreurs ,nous avons simplement utilisé des boucles if pour vérifier si les Codes données par le Client sont présents dans les fichiers que nous avons à notre disposition. Nous avons utilisé cette méthode car elle est facile à implémenter mais elle n'est pas très optimisée .

II- Comment on s'est organisé:

Notre méthode :

Git hub en guise de repo git.
VsCode en guise de plateforme.
Nous avons utilisé un codespace avec Git Hub pour travailler en commun. Ce fonctionnement nous a apporté pas mal de problème au début donc plus de temps pour se lancer mais plus nous le maîtrisons plus il nous était pratique.

Ensuite, nous nous sommes répartit la charge de travail équitablement:

Khalid	Jovan
Au début du projet, je me suis occupé de la création des pipes ainsi que le formatage de la chaîne de caractère qu'il faut envoyer en réponse. Pour tester nous avons commencé avec un seul client et plus tard nous sommes parvenu à mettre en place plusieurs clients. Par la suite, je suis passé au Serveur de données tout d'abord avec un seul serveur de données. Le problème qu'on avait au début c'est que le serveur renvoyait tout le menu à la place d'un seul article. Le problème fut vite réglé et j'ai pu passer à la deuxième étape qui est d'implémenter plusieurs Serveur. Application Cliente: -Formatage des Codes du Client en entrée -Création des pipes Client/Routage et Routage/Serv Données -Organisation pour respecter le protocole de communication Serveurs de Données: -Faire en sorte de pouvoir communiquer avec plusieurs serveur de données (dans notre cas à nous 2 serveur) -Parsing du message reçu du serveur routage contenant les Codes envoyés par le Client -Pouvoir lire les fichier.txt des lieux et menu afin de les renvoyer au client -Lecture de chaque fichier et récupération des Lieux et des Menus selon les codes reçu de la part du Client -Gestion des erreurs et envoie à l'application cliente selon le protocole de communication	Dans un premier temps : Confection du **Makefile** : - make all - make clean - la syntaxe automatique Dans un second temps : Les protocoles d'envoie et de reçu des messages dans les 3 interfaces (une méthode partiellement changé par la suite) Dans un troisième temps : Nous nous sommes naturellement redirigé vers une ou des entités chacun, pour ma part : **Serveur de Routage :** - Automatisation des pipes en fonction du fichier de configuration serveur - Gestion de la Table de routage aller et retoour - transmission des messages

III-Ce qui ne fonctionne pas:

On ne peut avoir que 2 Clients Max : Même si dans le serveur de Routage il est demandé de mettre le nombre de clients max, si on met 3 par exemple ça ne marchera pas. Dans l'application cliente on gère que les cas où le client a comme id 1 ou 2. Ceci aurait pu être changé en suivant la même méthode utilisée dans le serveur de Routage pour initialiser plusieurs pipes selon le nombre de clients.

La gestion d'erreur: Nous n'avons pas traiter toutes les erreurs avec des messages précis, on affiche des message simple du type: " Serveur Inconnu", " Erreur code menu"... La partie code pour gérer les erreurs est assez confuse et part dans tous les sens, on aurait pu mieux optimiser cela. Possiblement en utilisant stderr.

Multithreading: Nous n'avons tout simplement pas eu le temps d'aborder cette partie.

Filtre : Nous n'avons pas implémenté de filtre sur le menu. Tout d'abord il aurait fallu ajouter un nouveau paramètre pour le Client qui est les allergènes. Il aurait également fallu avoir un tableau de tous les allergènes. On aurait peut être pu implémenter cela au niveau du serveur de donnée lorsqu'on récupère l'article du menu avec un if qui vérifie si cet aliment est dans le tableau des allergènes.

Il est aussi nécessaire à certain moment de fermer le terminal avec un ctrl c et relancer l'opération lors d'une erreur de code par exemple. Exemple: Si l'utilisateur rentre le NumClient 3 il verra un message d'erreur qui dit : "Veuillez choisir un autre NumClient entre 1 et 2" Mais il sera toujours dans l'application cliente qui est en train de lire sur le Pipe avec le serveur de Routage. Ceci est probablement dû au while(1) dans le code que nous n'avons pas changé ou enlevé.

Communiquer avec plusieurs serveurs de données à la fois. En effet, on ne peut communiquer qu'avec un seul serveur de données à la fois.

IV- Comment aller plus loin:

Un des problèmes (ou plutôt inconvénient) de notre programme est qu' une fois une demande envoyée, il faut relancer l'application avec une nouvelle requête. Le serveur de Routage et le Serveur de donnée se ferment directement après avoir apporter la réponse au Client

Il aurait fallu développer ce point là, notamment en donnant la main au client d'entrer une nouvelle requête directement en stdin après avoir reçu la réponse de sa première requête afin de corriger ses erreurs sans avoir à relancer le terminal.

Nous aurions également pu implémenter un filtre comme cité précédemment.

La gestion des erreurs aurait pu être plus optimisée peut être avec l'utilisation de stderr mais nous n'avons pas vraiment explorer cette possibilité.

Il aurait été possible d’avoir réellement une multitude de clients en modifiant le code source de l’application cliente sous le même format que le Serveur de Routage en exploitant la structure Pipe.