CARTA Georges
NOWAK Axel
GALVES Quentin

RAPPORT PROJET SYSTÈME/RÉSEAUX

Sujet : Application client/serveur TCP/IP

Année :2019-2020

Licence 3 MIAGE
Semestre 5

Université Grenoble Alpes Antenne Valence

SOMMAIRE

I/ Introduction

Le but du projet est de créer une application client/serveur tcp/ip permettant à des utilisateurs de consulter des listes de trains. Cette liste de train contient un numéro, une ville de départ, une ville d'arrivée, les horaires de départ et d'arrivée, le prix et un champ optionnel de valeur « reduc » ou « suppl ». Puis un client se connecte sur un serveur et envoie une requête. Il a 3 choix qui s'offre à lui :

1. Soit il envoie une requête avec une ville de départ et d'arrivée le serveur lui renvoie une liste de trains satisfaisant sa requête.

2.Soit une requête avec une ville de départ, une ville d'arrivée et un horaire, le serveur renvoie un train correspondant à la requête.

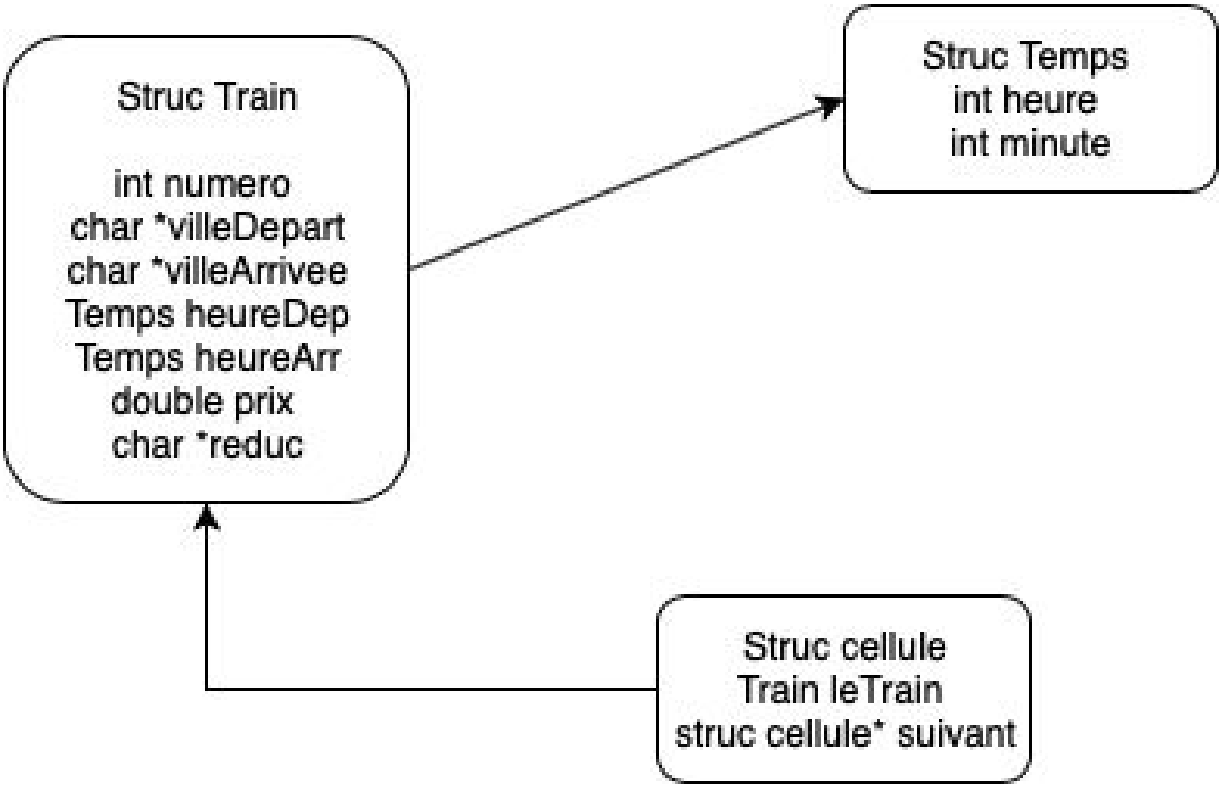
3.Soit la requête du client comporte une ville de départ une ville d'arrivée et une plage d'horaire, la requête renvoie la liste des trains possibles.

Puis si le client reçoit une liste de trains il pourra demander le trajet au meilleur prix ou le trajet de durée optimum. Dans la partie optionnel le client pourra interroger plusieurs serveurs, une partie administrateur qui lui permettra de voir les statistiques des requêtes par exemple : quel protocole est le plus utilisé ? Quelle est le trajet le plus demandé ?
Et enfin la partie « esthétique » comme la colorisation des résultats, …

II/ Explication des fonctionnalités

Un fichier texte est stocké sur le serveur contenant la liste de trains, au format suivant :

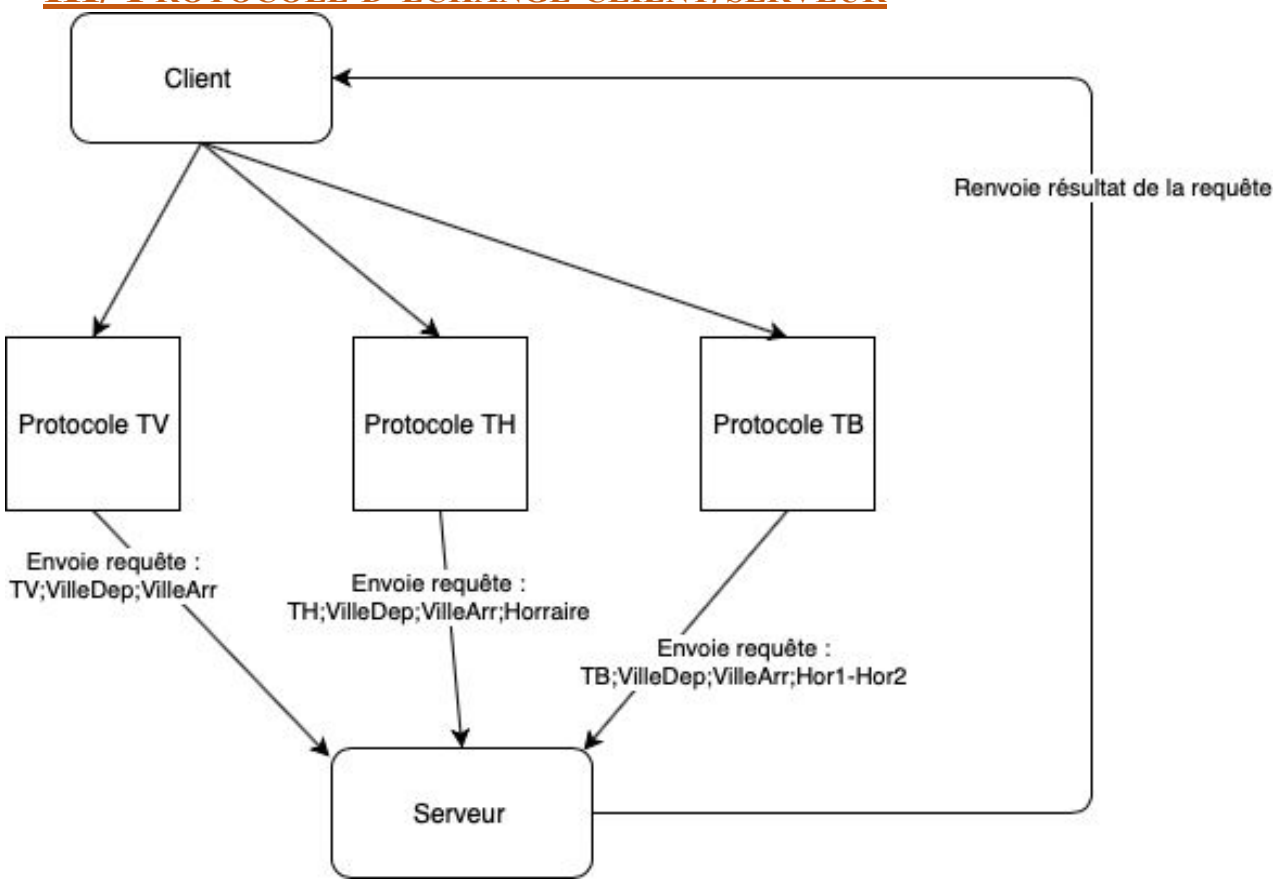
Numéro ;Ville de départ ;ville d'arrivée ;heure de départ ;heure d'arrivée ;prix ;reduc ou suppl



Tous les champs sont séparés par un « ; ». Nous récupérons ligne par ligne que nous stockons dans un tableau. Puis nous passons la ligne a un « strtok » pour la décomposition de la chaîne en plusieurs éléments. Ensuite nous remplissons une structure train en adéquation avec les éléments correspondants. Par exemple nous savons que le premier élément du tableau sera le numéro du train donc nous assignons cette valeur au champs numéro de la structure du train.

A la fin de cette méthode nous aurons une liste chaînée contenant tous les trains.
Nous avons choisi une liste chaînée de structure de train pour simplifier l'utilisation de notre liste et l'optimisation de la mémoire.

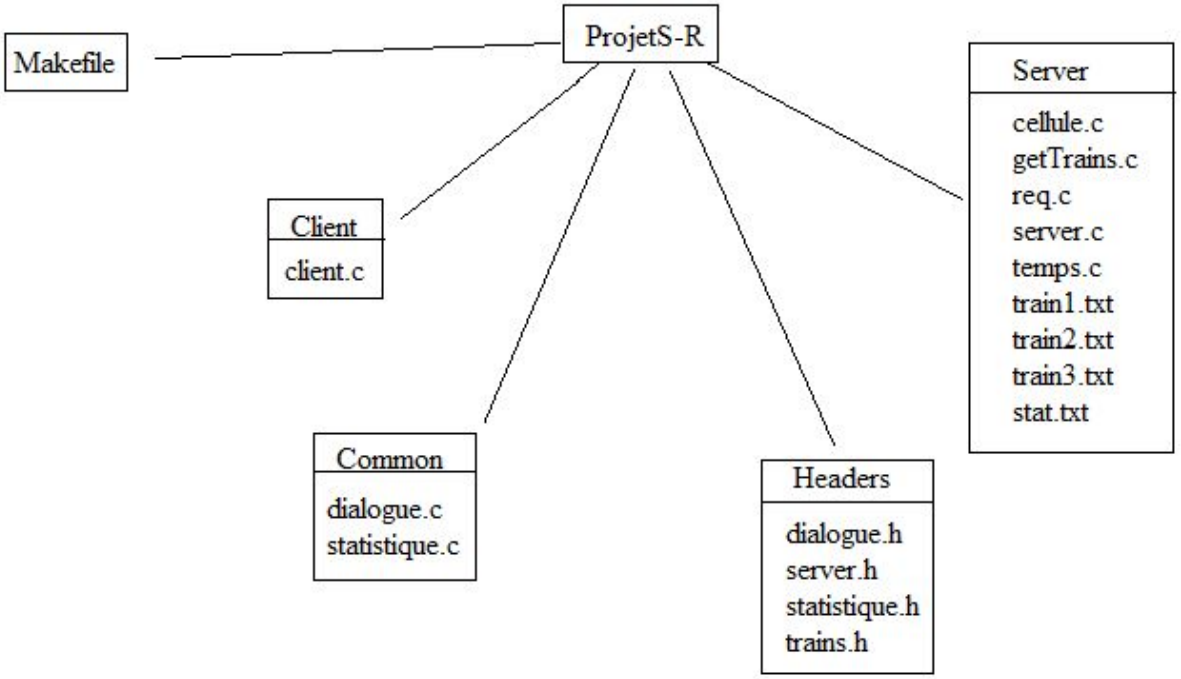
III/ Protocole d'échange client/serveur



Dès que le client se connecte il aura le choix entre ces 3 protocoles :

1. **TV** : Le client choisira le protocole TV S'IL veut la liste de tous les trains d'une ville de départ à une ville d'arrivée.
2. **TH** : Ici le client donnera une ville de départ, une ville d'arrivée et un horaire. Le serveur va renvoyer le train correspondant à ces critères
3. **TB** : Le client donnera une requête contenant une ville de départ, une ville d'arrivée et 2 horaires séparées par un « - »

Dès que le serveur reçoit la requête il la traite selon le protocole utilisé et renvoie-le ou les résultats.



IV/ ARCHITECTURE DE L'APPLICATION

Client :

- Client.c : créer la connexion avec les serveurs envoie la requête au serveur et récupère les trains correspondants.

Common :

- Dialogue.c : s'occupe du dialogue entre le serveur et le client.
- Statistique.c : sert à calculer et renvoyer au client (admin) les statistiques du server.

Server :

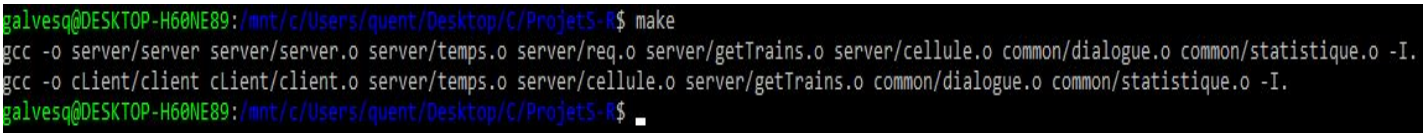
- Cellule.c : permet de créer une chaîne de train.
- getTrains.c : trie les trains et stock les bons selon le protocole utiliser.
- req.c : permet de traiter le train.txt et de stocker toutes les données.
- server.c : se connecte au client récupère ses requêtes et lui renvoie les bons trains.
- temps.c : comporte les fonctions en rapport avec le temps (comparaisons et transformation en string).

Headers :

- dialogue.h : définition des couleurs, des fonctions.
- server.h : définition des fonctions.
- statistique.h : définition des structures Stat CelluleT et des fonctions.
- trains.h : définition des structures Temps Train Cellule, et des fonctions.

V/Mini Manuel

Afin de compiler le projet il suffit de se mettre dans « /ProjetS-R$ » et de lancer « make » qui compile tout le projet :



Pour lancer un serveur il suffit de se rendre dans « /ProjetS-R$/server » et de lancer « ./server <n° serveur> <fichier texte>», avec en paramètre le numéro du serveur et le fichier texte dans lequel les trains sont stockés.

Pour lancer client il faut se rendre dans « //ProjetS-R$/client » et lancer « ./client $ » $ correspond aux numéros des serveurs.

Si on veut lancer le client sur les serveurs 1 et 2 par exemple, on fait « ./client 1 2 »

VI/ Fonctionnalités optionnels

A) Le multi serveur

Coté server :

On entre en argument du programme le numéro du serveur, ce qui nous permet de le récupérer et d'ouvrir le port et le fichier associé. Ceci nous permet de lancer plusieurs serveurs sur different fichier textes.
Les trains sont récupéré puis stocké pour être utilisé dans toutes les requêtes client.
A chaque connection le serveur crée un fils dédié au client.
Une fois connecté au client il récupère sa requête puis effectue le trie en fonction de paramètres.
Une fois le trie fini, le serveur envois un requête contenant le nombre de train puis envoie les trains eux même chacuns leur tour sous cette forme :

Numéro ;Ville de départ ;ville d'arrivée ;heure de départ ;heure d'arrivée ;prix avec la réduction ou le supplément appliqué

Côté client :

Les numéros des serveurs auquel on veut se connecter sont précisé au lancement de l'application. Ils sont ensuite récupéré et stocké dans un tableau juste après géré la partie admin. Les connections avec les serveurs sont initialisées les unes après les autres et stockées dans des tableaux.
Puis on demande à l'utilisateur les paramètres de la recherche à effectuer.
L'envoie et la réception des requêtes avec les serveurs sont aussi traités séquentiellement et les réponses sont stocké dans une liste chaînée de Trains.
On affiche ensuite les résultats et on demande à l'utilisateur s'il souhaite sélectionner faire un tri supplémentaire qui sera traité par l'application client.

B) L'ADMINISTRATEUR ET LES STATIQUES DE L'APPLICATION

Quand le client se connecte il aura le choix entre « C » et « A » s'il choisit C il sera considéré comme un client c'est-à-dire qu'il aura le choix entre les différents protocoles expliqués au-dessus sinon il sera considéré comme un administrateur. Pour le côté administrateur il devra rentrer un mot de passe pour pouvoir accéder aux résultats des statistiques en cas de mot de passe incorrect il sera déconnecté.

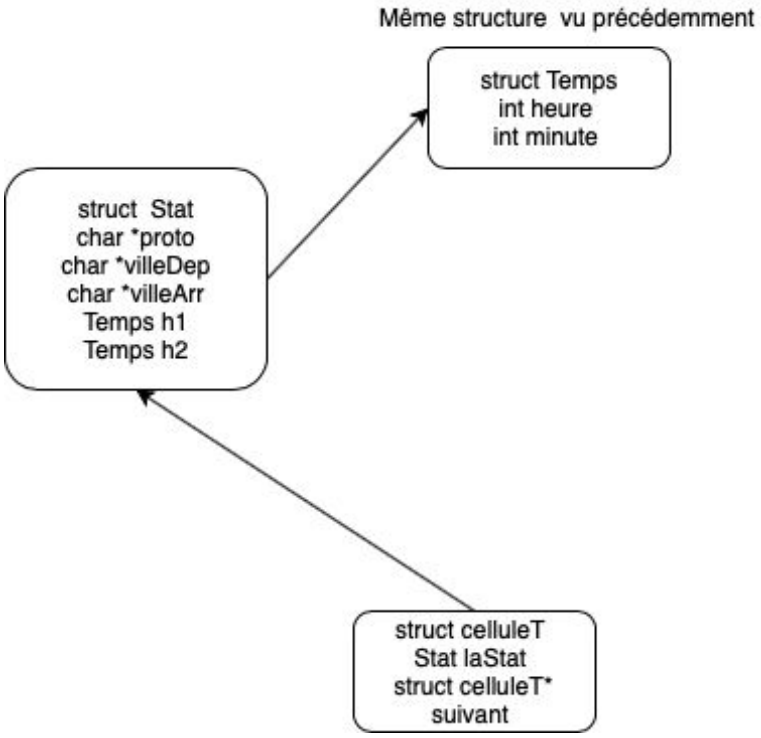
Pour le côté statistique nous avons procéder de la sorte :

1. Dès que le client envoie une requête au serveur le serveur écrit la requête dans un fichier texte (stat.txt). Puis nous procédons exactement comme pour traiter le fichier de la liste des trains.

   Les requêtes écrient sur le fichier stat.txt sont de la forme :

   Protocole-VilleDep-VilleArr-Horraire

Nous avons choisi des « - » entre chaque élément pour simplifier le code car si le client entre une plage d'horaire, c'est 2 horaires sont séparés par un tiret donc au lieu de faire un strtok pour décomposer la chaîne avec « ; » et un autre strtok pour « - » donc nous faisons un unique strtok de la requête avec « - ».



2. Nous avons créé une structure Stat pour procéder comme le remplissage de la structure train. Nous savons où chaque élément est situé dans le tableau. Donc par exemple l'élément proto de la structure Stat prendra le premier élément du tableau.
3. Nous avons créé différentes méthodes comme le max entre 3 entiers et la moyenne.
4. Puis nous comparons l'élément protocole de la struct Stat avec soit TV, TB, TH et nous stockons dans une variable pour voir combien de fois les protocoles sont utilisés. Puis le serveur renvoie à l'admin le protocole le plus utilisé avec sa moyenne.
5. De même le serveur renvoie avec le protocole le plus utilisé le trajet le plus demandés avec sa moyenne.
6. Enfin avec le protocole et le trajet, le serveur envoie si les clients demandent des trains soit le matin soit le soir.

C) L'AFFICHAGE DE L'APPLICATION

Nous avons défini des couleurs dans dialogue.h. Par exemple pour l'écriture cyan nous avons défini la variable cyan.

```
#define BOLDBLUE "\033[1;34m"
#define PURPLE "\033[35m"
#define BOLDPURPLE "\033[1;35m"
#define CYAN "\033[36m"
#define BOLDCYAN "\033[1;36m"
#define WHITE "\033[37m"
```

Puis pour appliquer ces couleurs nous faisons :

```
printf(CYAN "Veuillez indiquer le mot de passe : \n" DEFAULT);
fscanf(stdin, "%s", reponse);
```

Ici nous affichons « Veuillez indiquer le mot de passe : » en cyan et le default sert à remettre la couleur d'affichage par default

VII/ EXEMPLES

Exemple pour le côté administrateur :

```
galvesq@DESKTOP-H60NE89:~$ cd /mnt/c/Users/quent/Desktop/C/ProjetS-R/client
galvesq@DESKTOP-H60NE89:/mnt/c/Users/quent/Desktop/C/ProjetS-R/client$ ./client
Accès Administration(A) ou Client(C) ?
A
Veuillez indiquer le mot de passe
admin
Nous avons reçus aujourd'hui 43 requêtes
Le protocole TV est le plus utilisé avec 24 requete soit une moyenne de 55.814 %
Le trajet le plus recherché est Valence --> Grenoble avec 41 requete soit une moyenne de 95.349 %
La majorité des requêtes sont faites le matin
galvesq@DESKTOP-H60NE89:/mnt/c/Users/quent/Desktop/C/ProjetS-R/client$
```

Exemple pour le protocole TH :

```
galvesq@DESKTOP-H60NE89:/mnt/c/Users/quent/Desktop/C/ProjetS-R/client$ ./client 1
Accès Administration(A) ou Client(C) ?
C
Recherche avec ville seul(TV), avec une horraire (TH), avec deux horraires(TB) ? TH
Ville de départ : Valence
Ville d'arrivé : Grenoble
Horraire (XX:XX): 06:00
Voici Les trains disponibles  :
Train n° : 17568 | Départ (07:15) : Valence | Arrivé (08:32) : Grenoble | Prix : 17.40€
galvesq@DESKTOP-H60NE89:/mnt/c/Users/quent/Desktop/C/ProjetS-R/client$
```

Exemple pour le protocole TV et que le client veut le trajet le plus rapide

```
galvesq@DESKTOP-H60NE89:/mnt/c/Users/quent/Desktop/C/ProjetS-R/client$ ./client 1
Accès Administration(A) ou Client(C) ?
C
Recherche avec ville seul(TV), avec une horraire (TH), avec deux horraires(TB) ? TV
Ville de départ : Valence
Ville d'arrivé : Grenoble
Voici Les trains disponibles  :
Train n° : 17568 | Départ (07:15) : Valence | Arrivé (08:32) : Grenoble | Prix : 17.40€
Train n° : 17566 | Départ (06:45) : Valence | Arrivé (07:55) : Grenoble | Prix : 17.60€
Train n° : 17564 | Départ (06:15) : Valence | Arrivé (07:31) : Grenoble | Prix : 14.08€
Voulez vous le trajet le plus rapide(R) ? Le moins cher(P) ? Q pour quitter
R
Le train le plus rapide est :
Train n° : 17566 | Départ (06:45) : Valence | Arrivé (07:55) : Grenoble | Prix : 17.60€
galvesq@DESKTOP-H60NE89:/mnt/c/Users/quent/Desktop/C/ProjetS-R/client$
```

VIII/ Conclusion

Pour conclure, ce projet nous a permis d'en apprendre plus sur le langage C. Nous avons rencontré des difficultés surtout pour débugger.
Ce projet nous a permis de bien utiliser Git pour nos échanges et de voir comment un projet de groupe se fait (répartition des tâches, …).
Au cours de ce projet nous avons pu réaliser l'ensemble des besoins que le client souhaitait. Nous avons aussi eu le temps de programmer les besoins optionnels tels que le multi-serveur, les statistiques du serveur grâce à une connexion admin et l'affichage en couleur.