Entressangle Loïc, Avanzino Aurélien, Bahous Zakaria / L3 MIAGE

# Rapport de projet Système/Réseaux

# Sommaire:

Introduction, cahier des charges
I. Structure et idées concept
II. Interaction Client/Serveur
III. Programmes et protocoles
IV. Compilation et difficultés rencontrés
Conclusion .

Introduction, cahier des charges.

Ce projet rassemble nos connaissances de systèmes et réseaux, en effet il consiste en la réalisation d'une application client/serveur permettant aux utilisateurs la consultation des listes de trains voire même les choisir selon certains critères qui pourront être l'endroit de départ/destination ainsi que les heures de départ et d'arrivée. Il va donc falloir mettre en place une plateforme TCP/IP de gestion de trains et de trajets.

Cahier des charges :

-Le projet nécessite d'avoir un client et un serveur, un serveur qui gère les données de trains et le (ou les) clients qui ira l'interroger sur ces données. Il faut également que plusieurs clients aient la possibilité de communiquer avec le serveur simultanément.

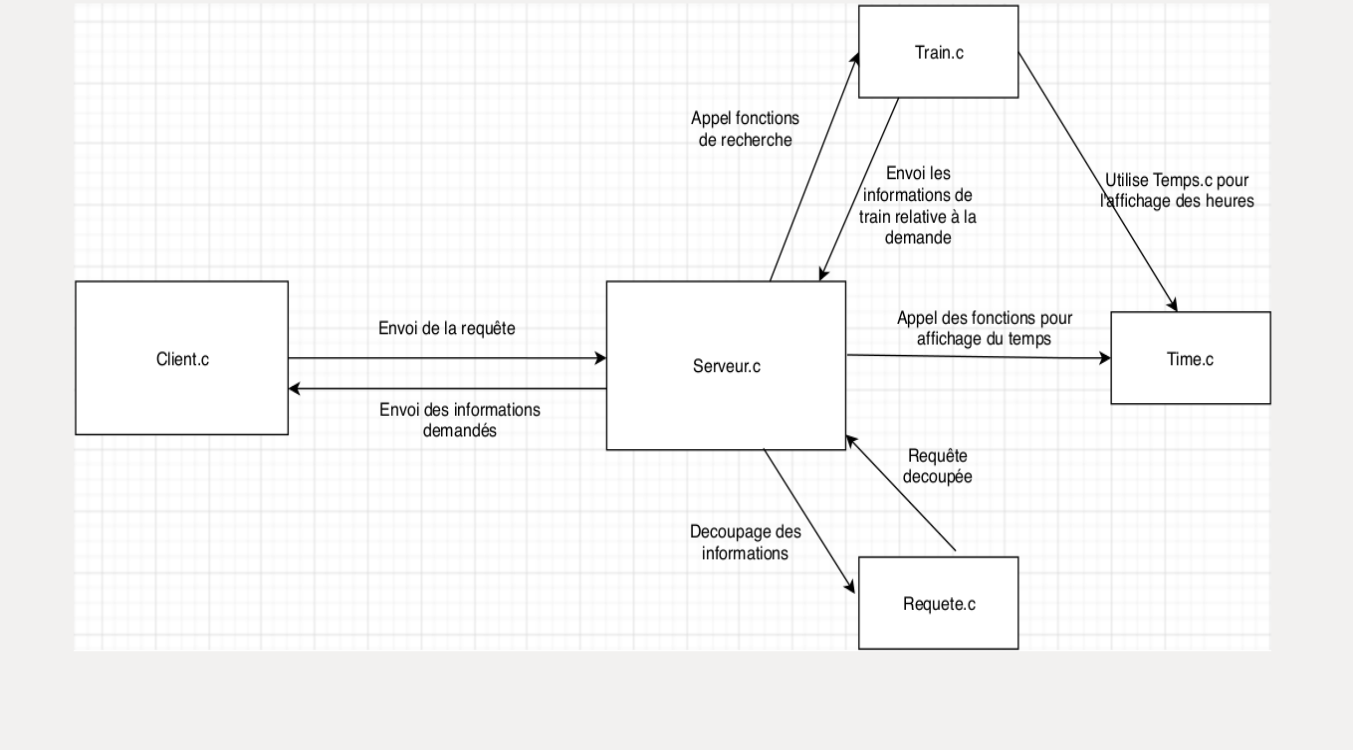
-Il doit être possible pour le client de pouvoir soumettre 3 types de formules de recherche de trajets et de trier les résultats par prix et ou durée de trajet.

- Mise en place des méthodes de gestion et d'affichage pour les trains et les temps.
- Mise en place d'une méthode pour le découpage de la requête

I. Structure et idées concept.

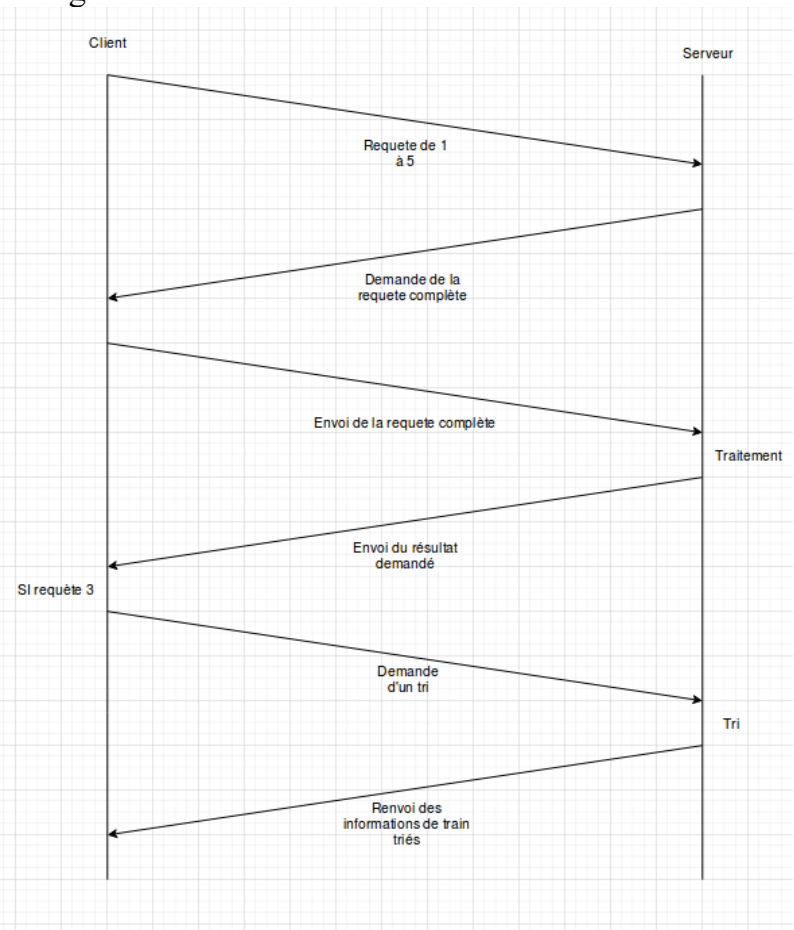
Le serveur représente le noyau du système. Celui-ci correspond à la partie « receive » c'est à dire qu'il doit être capable de traiter les requêtes du client.
Le client quant à lui est la partie « send », il envoie au serveur la nature de sa demande et les informations qui vont avec, les critères de recherche.  En effet le client émet une requête au serveur, puis le serveur envoi à son tour les spécifications de la requête au client, et demande au passage les paramètres de ses recherches. S'en suis échange et réponses jusqu'à ce que le client ai reçu les informations qu'il désire.
Cette structure permet un traitement entièrement basé sur une socket, et augmente grandement l'efficacité.
Pour ce qui est de notre interface le client une fois connecté au serveur aura une liste de choix numéroté de 1 à 4 pour la sélection des trajets de trains. Il pourra sélectionner celui qui l'intéresse en tapant directement son numéro afin d'émettre sa requête.
De son côté une fois qu'il a reçu la requête du client le serveur va récupérer les informations relatives à la demande pour ensuite lui envoyer. Le serveur renvoi alors un résultat ou un tableau de résultats selon la requête.
Le tri quant à lui est la seconde requête, accessible si plusieurs résultats ont été obtenus au préalable. Le client aura alors à choisir sur quoi le tri se fera entre durée, prix ou temps.

Voici la structure de notre application :



## II. Interaction Serveur/Client.

L'interaction serveur/client se fait a l'aide d'un socket Tcp/Ip comme vu ci-dessous.
Le client une fois connecté au serveur formule le type de la demande entre les 4 proposés. A ce moment le serveur détecte le numéro de demande et en fonction de cette dernière renvoi une lettre sur le socket. Selon le type de la demande cette lettre sera différente, ainsi coté client en fonction de la lettre récupéré il pourra alors envoyer les critères de sa demande sous une forme bien précise (DDDD AAAA HH:MM HH:MM) ou encore afficher les cas d'erreur si par exemple le client envoi au serveur une information qui ne respecte pas le format demandé ou encore si il entre un  mauvais numéro lors de la sélection du type de demande.
Le serveur initialise une structure de train dans laquelle on va appeler la méthode stockageTrain() afin d'avoir un tableau de tous les trains contenus dans Trains.txt.Une fois que le serveur a réceptionner les critères de recherche il va faire un découpage sur cette dernière et le stocker dans un tableau de char. Dans une autre structure Train on range le résultat de la méthode findTrain(table,requête) qui prend en paramètre la structure de Train initialisé avant et un char**. Enfin on stocke l'affichage en faisant appel aux méthodes d'affichage dans un char* (nous avons plusieurs méthodes adapté à la demande du client si celui-ci ne veut qu'un trajet ou plusieurs) . Pour finir le serveur envoi ce char* au client qui pourra le réceptionner grâce à un read et l'afficher.



## III. Programmes et protocoles.

Nous avons créé plusieurs fichiers de programmes, Requetes.c Time.c et Train.c et ceux-ci regroupent l'ensemble des méthodes permettant la gestion des requêtes client.

Explication de nos choix lors de la création de ces fichiers :

- Time.c :

- Permet de définit la structure temps composée de deux entiers définissant l'heure et les minutes.
- Création de fonction de calcul (*diffM et diffH*) pour observer l'écart entre deux horaires dans le cadre de la requête avec une plage horaire possible.
- Permettre une conversion simple et rapide entre un type char et un type temps via la création de la fonction *struct temps ctot(char *)*

- Train.c

- Permet de définir la structure Train composée de différents afin d'enregistrer l'ensemble des informations concernant un train :
    - int id : L'identificateur du Train
    - char depart[50] : Ville de départ du Train
    - char arrive[50] : Ville d'arrivée du Train
    - struct temps heureD : L'heure de départ
    - struct temps heureA : L'heure d'arrivée
    - float prix : Pris du train sans réduction
    - char RS[5] : Indique si le train dispose d'une réduction, d'un supplément ou non
- Permet un stockage plus aisé des trains dans un tableau et ainsi facilite la recherche et l'accès à ces derniers lors des différentes requêtes effectuées par le client
- Stockage de l'ensemble des fonctions de recherches des trains correspondant aux différentes requêtes envoyées par l'utilisateur client.
- Le fichier permet, à l'aide d'un tableau alloué de façon dynamique, le stockage de la liste des trains enregistrée dans un fichier .txt
- Il contient également l'ensemble des fonctions permettant la transmission au serveur sous la forme d'un *char* * du train ou de la liste de train satisfaisant la requête client.
    - findTrain(struct Train * C, char **D) retourne le train correspondant à la requête
    - bestTrain(struct Train *C, int choix) retourne le train au meilleur prix ou partant le plutôt en fonction du choix de l'utilisateur et de la requête précédente.
    - findLTrain(struct Train *C, char **D) renvoie l'ensemble des trains partant dans la plage horaire indiquée par le client
    - findAllTrain(struct Train *C, char **D) retourne l'ensemble des trains partant d'un point et arrivant un point B

- Requêtes.c

- Contient toutes les fonctions permettant l'ensemble des actions possibles sur la requête effectuée par l'utilisateur tel que :
    - replace(char *C) : permet le remplacement du \n par un \0 pour ainsi toute erreur lors de comparaisons entre différente chaîne de caractères.
    - découpage(char *C, char **D) qui effectue un découpage de la commande l'utilisateur client pour ensuite être utilisée dans les fonctions requêtes de Train.c
    - correct(char ** D, int fct) qui aurait du être utilisée afin de vérifier que la commande de l'utilisateur soit correctement écrite. Mais par manque de temps nous n'avons pas pu l'intégrer dans le main de notre programme

Ces différentes répartitions des codes dans différents fichiers.c nous as permis également de scinder le travail rapidement et efficacement et ainsi permettre l'avancement du projet sur plusieurs fronts en même temps (notamment la séparation système et réseau).

De plus nous avons ajouté l'utilisation d'un menu afin d'aider l'utilisateur Client.
Ce dernier permet de communiquer simplement entre le programme Client.c et Serveur.c via l'utilisation d'un switch/case.

## IV. Difficultés rencontrés

- ➔ Problème d'allocation et libération de l'espace mémoire lors de la création de tableau dynamique causé par des tailles pas assez grandes et une allocation de mémoire pas assez importante
- ➔ Problème de transmission des informations entre le serveur et le client sur les read() et les write() à certains moments les informations n'étaient pas reçu
- ➔ Adaptation du code des différents fichiers .c tels que train afin de pouvoir être utiliser par le serveur. En effet certaines méthodes de Train.c etaient en void il a donc été nécessaire de les modifier afin qu'elles renvoient un résultat utilisable par le serveur
- ➔ Nous avons rencontré des problèmes de compilation on a donc mis en place plusieurs Makefile, un pour le serveur qui compilera avec Train.c, Requête.c, Time.c et un autre pour la compilation du client. Un troisième Makefile situé à la racine du projet va lors de son exécution appeler les 2 autres pour une compilation complète en une seule fois.
- ➔ Problème sur les pc de la fac par rapport au port qui étaient inaccessibles, il nous fallait modifier le numéro de port du serveur et du client.

## Conclusion et ce que l'on aurait aimé ajouter

Ce projet nous a permis d'en apprendre plus sur la création de socket et l'interaction entre client et serveur mais aussi sur le langage C en général. Malgré les problèmes rencontrés nous sommes plutôt satisfaits du résultat. Nous n'avions pas eu le temps de tout faire notamment pour le cas d'erreur lors de l'entrée d'une requête si le client entre une mauvaise requête, on suppose au début que toutes les requêtes sont entrées correctement. Nous n'avons également pas eu le temps de faire la récolte de statistique.