Rapport Projet Système & Réseau

8 Décembre 2017

Participants :

VINCENT Axel,
BERGER Pierre,
CHRETIEN Michel,
GRAVER William.

Objet du rapport :

Ce projet s'inscrit dans le cadre du cours " Système" et "Réseau" de la licence 3 MIAGE de l'Université Grenoble Alpes

Table des matières :

1. SUJET

Le but de ce projet est de réaliser une application client/serveur TCP/IP permettant à des utilisateurs de consulter des listes de trains disponibles entre deux villes données, ainsi que de proposer des options supplémentaires qui permettent de les trier selon différents critères. Les informations sur les trains existants seront rassemblées sur un serveur, que les clients interrogeront selon les besoins des utilisateurs.

2. ORGANISATION

En considérant la difficulté de réaliser un projet à plusieurs, nous avons décidé dès le début du développement de définir certaines règles, de ce fait nous avons établi des conventions de programmation dans le but d'obtenir du code maintenable et compréhensible de tous. Nous avons alors défini une standardisation du code et mis en place un logiciel de gestion de version afin d'obtenir un cadre de travail optimal tout au long du projet.

a. Standardisation

Dans le but de rendre le code le plus lisible et le plus compréhensible possible pour l'ensemble du groupe et pour le corps enseignant, nous avons écrit un fichier d'exemples avec les conventions de nommages à utiliser. Ceci nous a permis de comprendre quelles étaient les macros, les variables. Au niveau de l'indentation nous avons décidé d'utiliser le style Allman et pour la typographie des variables nous avons utilisés le ***lower camel case*** avec de plus des noms de variables les plus explicites possible. Voici le fichier d'exemple appliqué tout au long du projet :

```c
/**
 * @file main.c
 * @date 4 Dec 2017
 * @brief Fichier exemple des conventions
 *
 * @see ../headers/train.h
 */

 /**
  * @fn int main(int argc, char *argv[])
  * @brief Point d'entrée du programme
  * @param argc Le nombre d'argument de argv.
  * @param argv Tableau d'argument fournit au programme
  * @return int Resultat de l'éxécution
  * @TODO Expliquer à quoi sert un TODO.
  */
int main(int argc, char *argv[])
{ // Les accolades sont toujours à la ligne
   if (2 == 6) // Après un if/else/while, un espace avant la condition. Dans la condition entre chaque élémen
   {
    int monChat = oui; // Variable premiere lettre en minuscule puis le mot suivant commence par une majuscul
   }
   return 0;
}


 /**
  * @fn int PetiteFonction()
  * @brief Fonction qui renvoie 42
  * @return int Renvoie le sens de la vie
  * @TODO Trouver le sens de la vie
  */
int petiteFonction() // Le nom des fonctions commence par une premiere lettre en minuscule puis le mot suivan
{
   return 42;
}
```

Annexe 1 : Document de standardisation du code

De plus afin de permettre une compréhension aisée des fonctions développées nous avons décidé de mettre en place le générateur de documentation Doxygen, ainsi que

d'appliquer une certaine rigueur quant à la rédaction de commentaires, c'est à dire que nous nous sommes efforcés d'insérer un commentaire dès que nécessaire.

b. Gestion du projet

Afin de mener à bien ce projet en équipe nous avons décidé au lancement de mettre en place un gestionnaire de version, git. Celui-ci nous a permis de pouvoir développer simultanément différentes fonctions du serveur et en cas de conflits entre les différents développements en cours, de les résoudre plutôt aisément. De plus étant décentralisé, git permet une certaine portabilité du projet, de manière à pouvoir le transporter facilement entre nos différents espaces de travail personnel et universitaire. Ce projet est disponible à l'adresse suivante : https://github.com/elvnct/AlloHouston.

Pour gérer le mieux possible le projet nous avons décidé de mettre en place une TODO list écrite sur un tableau et visible de tous afin de savoir ce que fait chaque membre du groupe. Ceci nous a permis de ne pas nous poser de question sur quelle tâche réaliser après en avoir fini une première. Tous les jours nous mettions à jour la TODO list afin de voir ce qui était réalisé la veille et ce qu'il y avait à faire le jour même afin de se tenir au courant de l'avancement.

3. RÉALISATION

Nous allons dans cette partie vous détailler la façon dont nous avons conçu le projet et la manière dont nous l'avons fait.

Au début du projet nous nous sommes mis d'accord conjointement afin de définir la manière de développer le projet et quelles fonctionnalités nous voulions implémenter. Nous avons mis un point d'honneur à mettre en valeur l'esthétique de notre solution afin de la rendre la plus agréable possible à utiliser comme si elle était mise plus tard en production. Nous avons aussi pris la décision de développer des fonctions "outils" modulables au maximum pour simplifier le code et accélérer le développement.

a. Le client/serveur TCP/IP

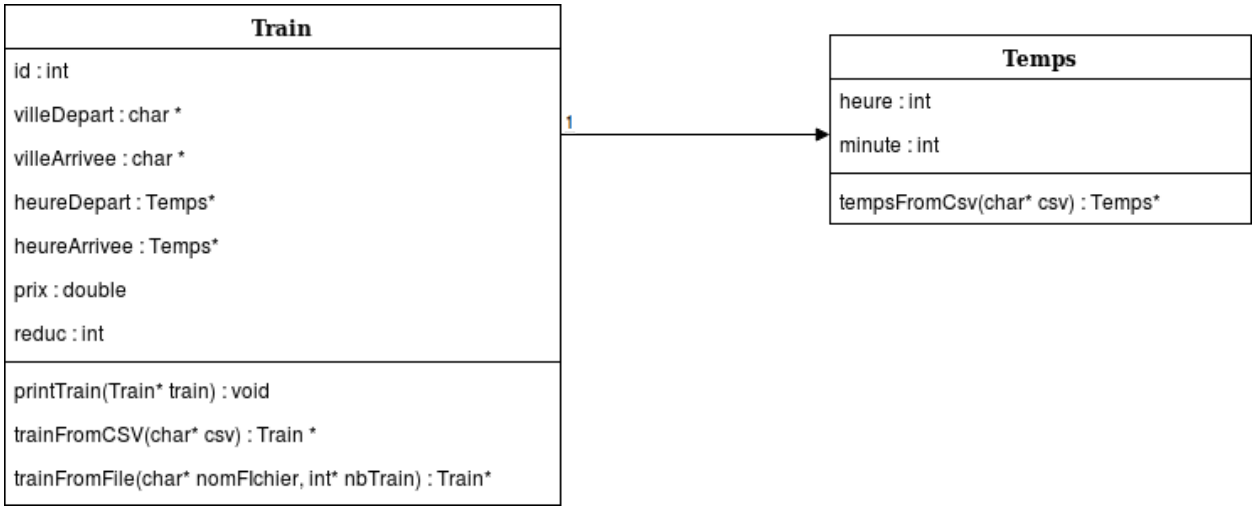
Dans un premier temps nous nous sommes attachés à développer le noyau du serveur TCP/IP. Cette partie a été réalisée ensemble afin que chacun interprète le fonctionnement fondamental du client et du serveur. Nous avons surtout essayé de permettre la multi-connexion sur le serveur, ainsi que permettre la connexion grâce au hostname du serveur. Cette partie nous a semblé très importante puisqu'elle permet une vision réaliste de la façon dont fonctionne un serveur (comme un serveur web, où un serveur ftp).

Par la suite nous avons essayé de minimiser le plus possible l'intelligence côté client, afin de reproduire les conditions réelles d'un client TCP/IP tel un client HTML, ou même un minitel. Cela signifie que le client ne fait que se connecter au serveur, puis recevoir et transmettre des informations au serveur. Il ne procède à aucune vérification sur les entrées de l'utilisateur, puisqu'elles sont côté serveur.

b. Requêtes sur le fichier

Dans cette phase du développement nous avons dans un premier temps créé un fichier de test pour la récupération du fichier d'entrée et des différentes requêtes réalisées sur les données de ce fichier. Ceci a été fait afin de permettre la compilation du projet principal peu importe l'avancement de cette partie.

Nous avons ensuite décidé de définir une structure afin de faciliter le traitement du fichier fourni. Celle-ci est définie comme présenté ci-dessous :



Annexe 2 : Diagramme UML de la structure Train utilisant une structure Temps

Puis nous avons défini des fonctions permettant le traitement de ce modèle de données. La première "printTrain()", permet d'afficher toutes les informations concernant un train donné en paramètre de cette fonction. "trainFromCsv()" permet à partir d'une ligne du fichier Trains.txt, de retourner un train, "trainFromFile()" utilise cette fonction à chaque ligne de train lue dans le fichier .txt pour créer une liste contenant tous les trains. Cette fonction est appelée à chaque connexion d'un nouveau client afin d'être sûr que les données sont bien à jour. Ce choix à un intérêt dans le cas où l'on peut ajouter des nouveaux trains ou si un train est en retard.

Enfin nous avons développés les fonctions permettant de satisfaire les requêtes devant être réalisées pendant l'échange avec le client. Dans un premier temps nous avons pensé utile que ces fonctions renvoient un Train ou une liste de Train. Après réalisation de la partie dialogue client/serveur nous nous sommes vite rendus compte que ce retour de liste n'avait pas grande utilité puisque nous construisons la chaîne de caractère à l'intérieur de la fonction de recherche.

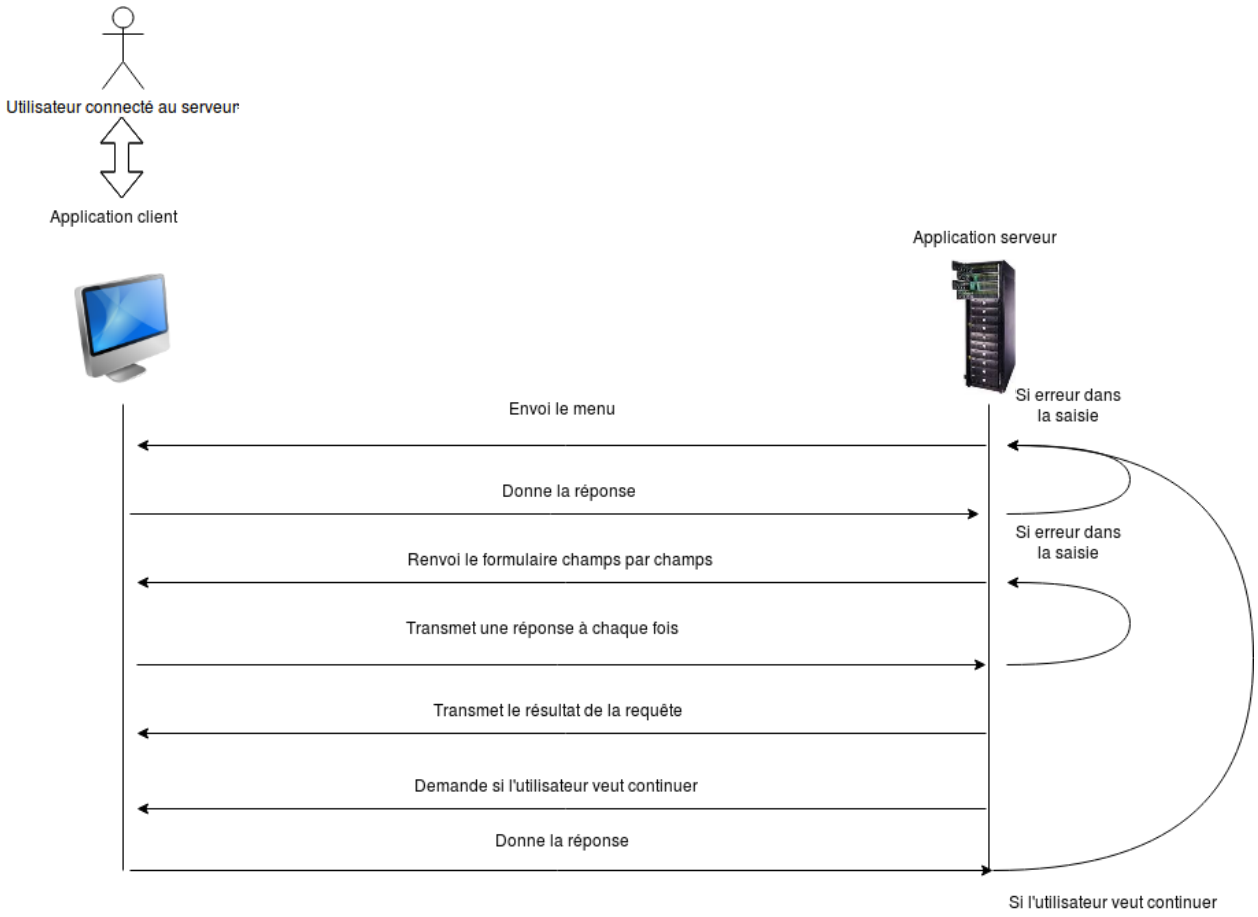
c. Interaction client/serveur

Dans cette partie nous n'évoquerons pas la connexion entre client et serveur puisque nous avons détaillé cette partie du projet précédemment mais nous évoquerons la manière dont sont transmises les informations entre le client et le serveur.

Le serveur commence par envoyer le menu au client, ensuite en fonction de la réponse que transmet le client, le serveur transmet champs par champs le formulaire permettant la recherche du/des trains. Après transmission de la réponse par le client, le serveur vérifie l'intégrité de la saisie et erreur si elle n'est pas correcte, celles-ci sont les suivantes :

- Vérification du format de l'horaire,
- Vérification que l'heure de départ est inférieure à l'heure d'arrivée,
- Vérification si la ville existe dans le fichier,
- Suppression des espèces,

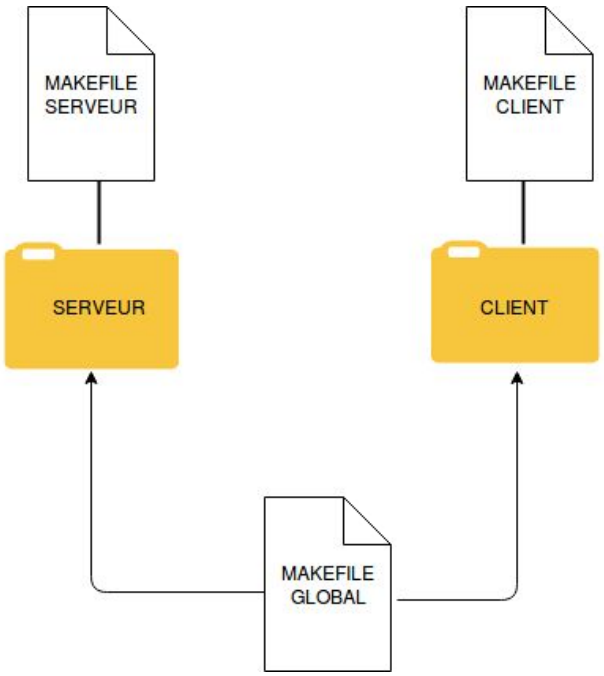
Après la saisie correcte des informations, nous affichons la liste des trains correspondant aux critères ou un message indiquant qu'aucun train n'a été trouvé. Enfin nous demandons au client s'il veut continuer de naviguer dans l'interface ou s'il veut quitter. S'il veut quitter, le serveur tue son fils alloué au service du client. Le schéma présenté ci-dessous résume ce fonctionnement :



Annexe 3 : Timeline de l'interaction client/serveur

4. COMPILATION

Chaque partie de notre projet est présent dans son dossier respectif comme indiqué dans les spécifications indiquées dans le sujet. Nous avons pris la décision de mettre en place un makefile global qui compile le client et le serveur en même temps en appelant les makefile présents dans chaque dossier. Grâce à un “cd” et à la macro $(MAKE) on appelle les différents makefile présents dans chaque partie indépendante du projet.



Annexe 4 : Schéma du makefile global

5. RÉTROSPECTIVE

Ce projet a été un réel défi pour nous tous car le travail en équipe est un processus difficile à mettre en place et à faire perdurer. Cependant la cohésion dans l'équipe a été un réel atout, de part l'implication importante de chaque membre du groupe. Concernant le projet, nous avons notamment apprécié le fait que le projet soit développé en seulement une semaine car cela nous a permis d'être focalisés seulement sur cette tâche. En conclusion ce projet a été très formateur pour nous puisqu'il nous a permis de découvrir et mettre en pratique le travail en équipe. De plus il nous a permis d'approfondir nos connaissances dans le langage C ainsi qu'appliquer nos connaissances en réseau quant à la communication TCP/IP.