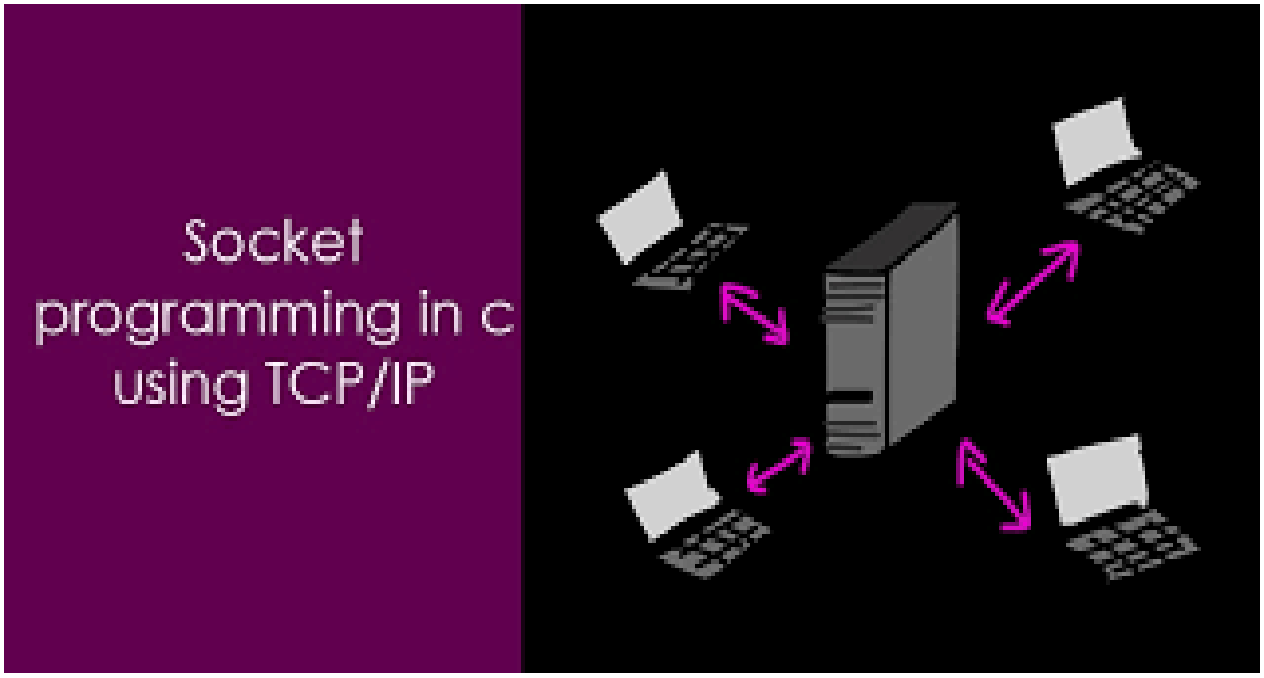
Programmation de Sockets en C avec TCP/IP

Systèmes & Réseaux L3 MIAGE - 2023/2024



Réalisateurs:

- **OURZIK Jugurta**
- **MIEL Nils**
- **DUMAS Loïc**
- **GREGOIRE Teddie**

SOMMAIRE :

1- Introduction

Le projet consiste en la réalisation en C d'une application **client/serveur TCP/IP** simplifiée basée sur la communication avec les sockets.

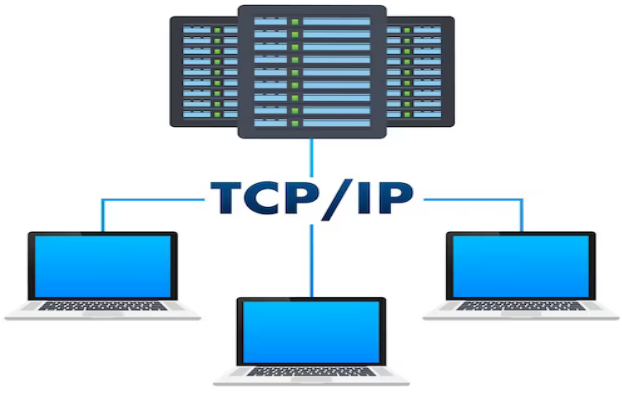
Nous allons réaliser une application client/serveur permettant à des utilisateurs de consulter des listes de trains disponibles entre deux villes données, voire de les sélectionner suivant des critères comme l'horaire de départ, un meilleur prix et l'optimalité du trajet.

Les informations sur les trains existants seront rassemblées sur un serveur, que les clients interrogeront selon les besoins des utilisateurs.

Les attentes du projet à réaliser seront de mettre en œuvre une communication entre processus via les sockets, la configuration du serveur et du client, la mise en place d'un protocole d'échange client/serveur et leur synchronisation.

L'objectif du projet consiste à comprendre le fonctionnement du protocole TCP/IP

dans les réseaux (à travers les sockets) et la gestion de processus en système pour établir la communication entre le client et le serveur une fois que la connexion a été acceptée.



2- Conception logicielle globale:

- 2.1- Définition de l'objectif : réaliser en C une application **client/serveur TCP/IP** simplifiée sur la communication avec les sockets.Autrement dit, faire communiquer deux processus en l'occurrence un client et un serveur pour répondre aux requêtes du client.

- 2.2- Planification :
    - 2.2.1- Choix de l'architecture du projet (Ressemble au pattern Model-View-Controller)

      Ici c'est vrai on n'a pas les views mais on peut considérer l'affichage comme un **view**, cela dit depuis le client (entrée standard) l'utilisateur envoie des données (ville de départ, ville d'arrivée et un horaire par exemple) suivant un **modèle** de données (le body de la requête contiendra :string, string, string) correspondant aux données d'entrée.Un **contrôleur** va ensuite gérer cette requête et dans ce cas il s'agit des fonctions associées à chaque interrogation dans le fichier sncf.c (**Le serveur**) et cela en interrogeant une pseudo base de données (le fichier sncf.txt), par la suite il renvoie le modèle construit comme réponse au client.

```
.
├── bin
├── doc
│   └── sncf.txt
├── headers
│   ├── client.h
│   ├── serveur.h
│   └── sncf.h
└── sources
    ├── client.c
    ├── mainClient.c
    ├── mainServeur.c
    ├── makefile
    ├── serveur.c
    └── sncf.c
4 directories, 10 files
```

- 2.2.2- Configuration du serveur :
  - Création de la socket d'écoute (primitive socket)
  - Attachement de la socket d'écoute à une adresse IP et à un port TCP. (primitive bind)
  - Mise en écoute du serveur. (primitive listen)
  - Attente de connexion dans une boucle infinie.(primitive accept)
  - Création d'un processus fils pour le client connecté. (primitive fork)
- 2.2.3- Configuration du client
  - Création de la socket du client (primitive socket) + définition d'un port de connexion et un host (primitive gethostbyname).
  - Connexion du client au serveur. (primitive connect).
  - Dialogue avec le serveur.
- 2.2.4- Mise en place du protocole d'échange.

  Pour synchroniser la communication entre le serveur et le client on a décidé d'adopter un protocole suivant les règles suivantes:

  - Transmissions multiples (comment ?)
    - Avant chaque transmission de donnée, sa taille en octet est envoyée en premier suivi de la donnée elle-même en suite d'octets. Exemple:

```
send size of the departure city
send departure city
send size of arrival city
send arrival city
```

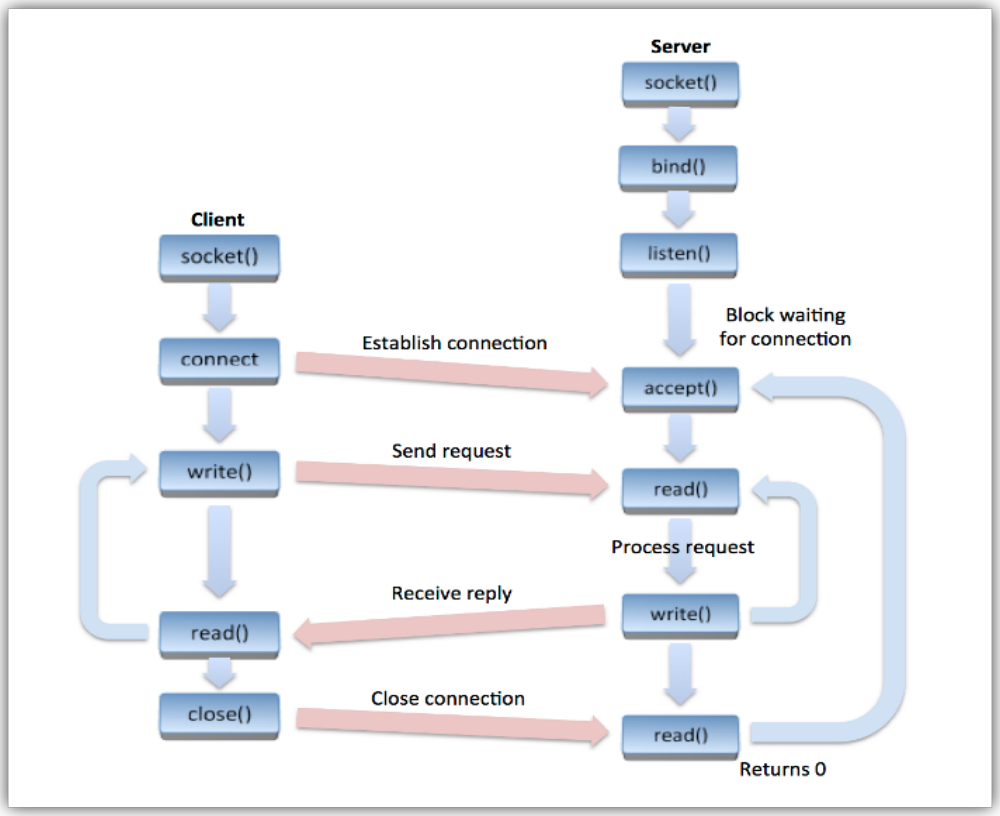
    - À la réception des données de la part du serveur, on fait pareil, on lit d'abord la taille suivie de la données elle-même.

```
read size of departure time
read departure time
read size of arrival city
read arrival city
```

- Choix du modèle des données.
  - Les données à transmettre suivent simple un modèle: Exemple

- N° du train : Int
- Ville de départ : String
- Ville d'arrivée : String
- Horaire de départ : dd:dd

- 2.3- Phase de test ; (Compilation et makefile) :
  - Modularisation (comme indiqué dans l'architecture au dessus)
  - Makefile
  - Tests unitaires pour chaque fonctionnalité implémentée.
    - Vérification des résultats de la fonctionnalité
  - Tests de robustesse pour les fonctionnalités implémentées.
    - Vérification des formats de données.
- 2.4- Phase de mise en production :
  - Le serveur est lancé et est prêt à gérer les requêtes du client.

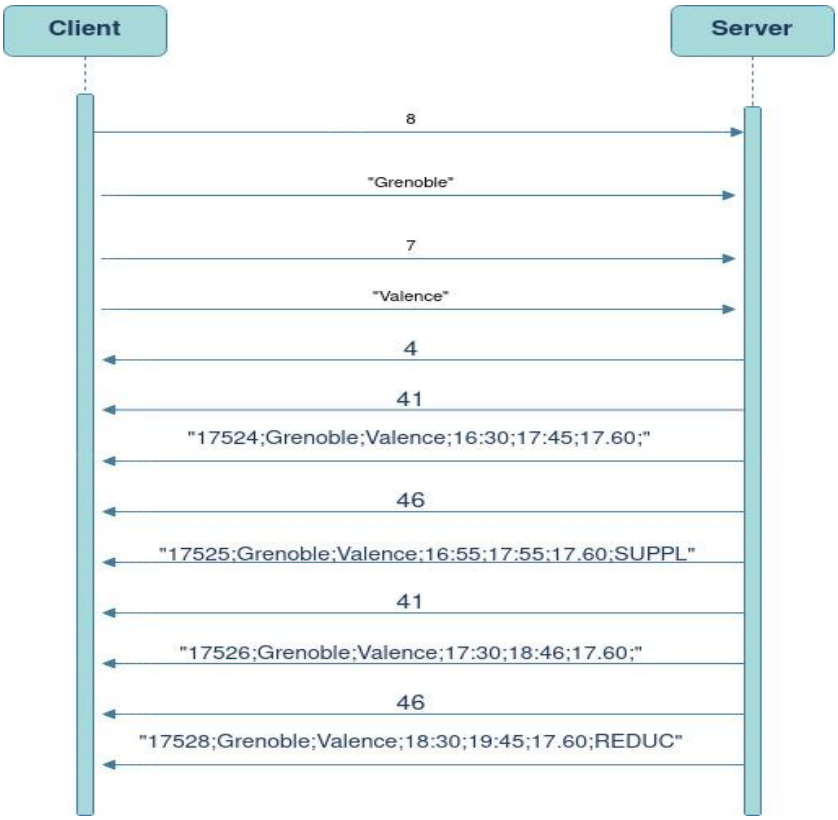


3- Protocole d'échange client/serveur

Un protocole d'échange de données traite tous les types de données quel que soit le format des messages échangés. Le protocole est un ensemble de règles formelles qui régissent :

- Le séquençage des flux d'octets ;
- La temporisation entre les émissions et les réceptions ;
- Le contrôle des erreurs à travers le réseau.

Ci-dessous le diagramme de séquences illustrant la définition de notre protocole d'échange et l'ordre de transmissions des données entre le client et le serveur:



Traiter les requêtes de l'utilisateur en fonction de son choix d'option (0,1,2 ou 3) et du protocole d'échange décidé que le serveur et le client utilisent pour ces multiples transmissions

- si option==0 :
    - Voir tous les trains (PAS DE LECTURE POUR D'AUTRES INFOS)
- sinon si option == 1
    - Obtenir le train par ville de départ, ville d'arrivée et heure de départ données
        - lecture en plusieurs fois et avant chacune précédée de la lecture de la taille de chaque donnée
            - lire la taille de la ville de départ
            - lire la ville de départ
            - lire la taille de la ville d'arrivée
            - lire la ville d'arrivée
            - lire la taille de l'heure de départ
            - lire l'heure de départ.
- sinon si option == 2
    - Obtenir le train par ville de départ, ville d'arrivée et créneau horaire de départ donnés
        - lecture en plusieurs fois et avant chacune précédée de la lecture de la taille de chaque donnée.
            - lire la taille de la ville de départ
            - lire la ville de départ
            - lire la taille de la ville d'arrivée
            - lire la ville d'arrivée
- sinon si option == 3
    - Prendre le train par ville de départ et d'arrivée donnée avec le meilleur prix ou la durée de trajet optimale
        - lecture en plusieurs fois et avant chacune précédée de la lecture de la taille de chaque donnée.
- sinon option invalide

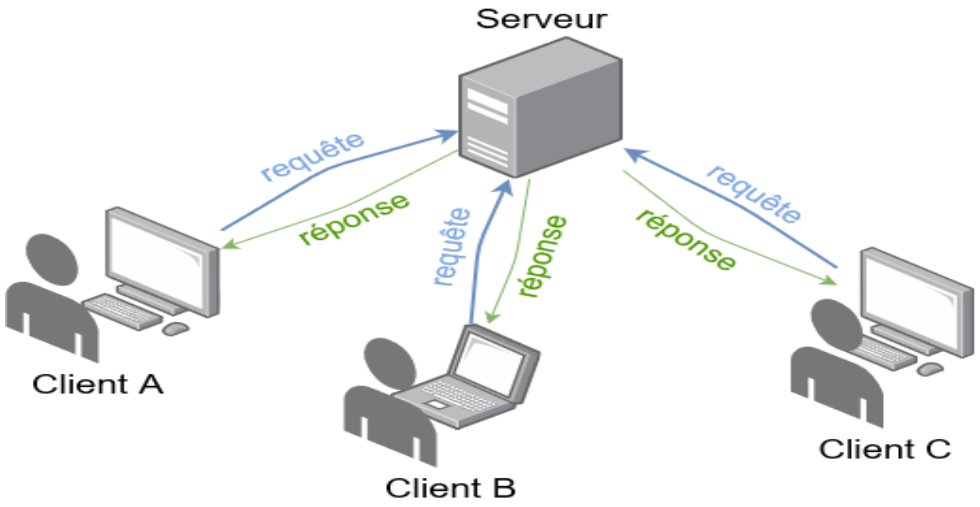
4- Aspects techniques de la mise en œuvre du protocole d'échange client/serveur dans l'application

- ☐ D'un point de vue technique, la mise en place de ce protocole d'échange entre le client et le serveur assure la **synchronisation** de ces deux derniers en:
    - Permettant de savoir la taille de données à recevoir
    - Permettant de savoir à quel moment les envoyer
    - Permettant de savoir.

5- Les actions possibles de l'utilisateur sur l'application

Le serveur prend en charge un ensemble de requêtes concernants les trains à savoir:

- ☐ Consulter la liste de tous les trains de la journée
- ☐ Consulter le train correspondant à une ville de départ, une ville d'arrivée et un horaire de départ.
- ☐ Consulter le train correspondant à une ville de départ, une ville d'arrivée ,un horaire de départ et une tranche horaire.
- ☐ Consulter le train correspondant à une ville de départ, une ville d'arrivée avec le meilleur prix ou ayant le trajet optimum.

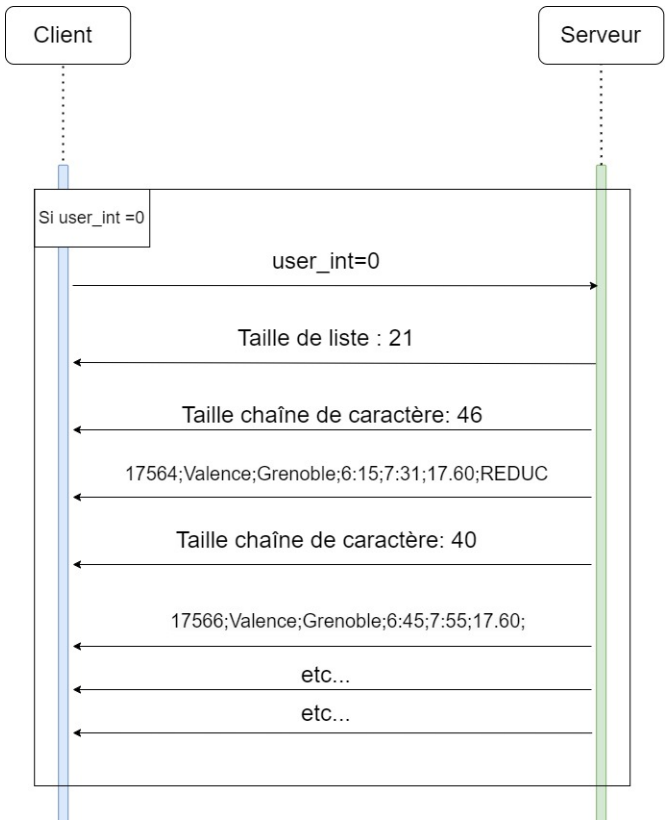


Remarque: Concernant la deuxième fonctionnalité implémentée, au cas où aucun train ne serait trouvé pour l'horaire donné, le prochain train sera renvoyé et au cas où l'horaire donné correspond à celui du dernier train de la journée (un peu tard), le premier train du lendemain sera renvoyé.

6- Explication des fonctionnalités implémentée

☐ Consultation de la liste des trains de la journée

Pour consulter la liste de tous les trains de la journée, il faut choisir l'option 0 dans le menu principal. Cela invoque la fonction getAllTrainsFromTheServer qui prend une socket de service en paramètre et qui transmet l'option choisie au serveur.



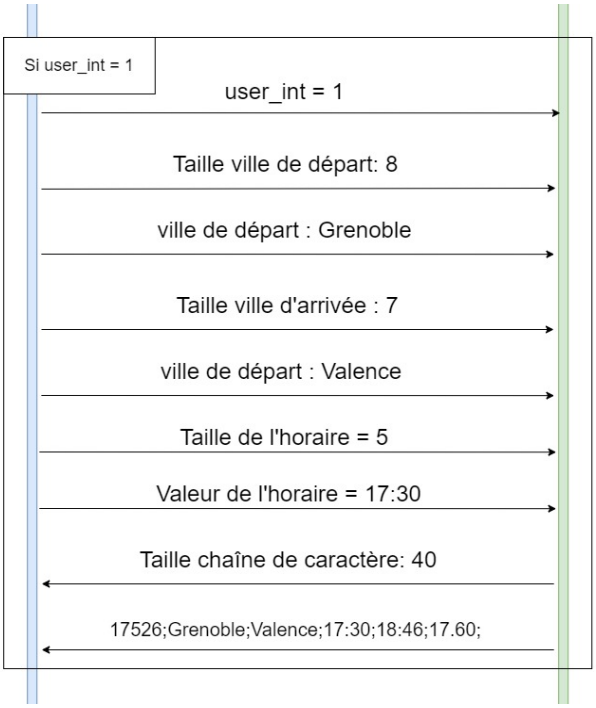
Le serveur lui effectue une recherche de tous les trains dans le fichier en paramètre (en l'occurrence le fichier sncf.txt) grâce à la fonction getAllTrains avant d'envoyer les données trouvées suivant le protocole d'échange décrit ci-dessus.

Le contrôleur getAllTrains ouvre le fichier en question (une pseudo base de donnée) et fait la recherche.Par la suite, il construit une liste de trains (liste de chaînes de caractères) et la communique à la fonction sendListeTrains. Cette dernière transmet en premier la longueur de la liste au client, ensuite en la parcourant elle transmet à nouveau la longueur de chaque chaîne de caractères (le train) et enfin le train lui-même.

Le client à sa réception de la réponse, il lit d'abord la longueur de la liste et boucle autant de fois pour lire la longueur du train pour pouvoir lire par la suite la chaîne exacte correspondant aux données du train et afficher chaque train sur la sortie standard.

☐ Consultation du train correspondant à une ville de départ,à une ville d'arrivée et à un horaire de départ.

Pour consulter le train correspondant à une ville de départ, à une ville d'arrivée et à un horaire de départ, il faut choisir l'option 1 dans le menu principal. Cela invoque la fonction `getTrainbyHourAndCityFromServer` qui va prendre en paramètre une socket de service et qui transmet au serveur l'option choisie. Elle envoie également au serveur les données rentrées par l'utilisateur avec les fonctions `askAndSendDeparture`,`askAndSendArrival` et `askAndSendTime` qui demandent à l'utilisateur de rentrer des données.



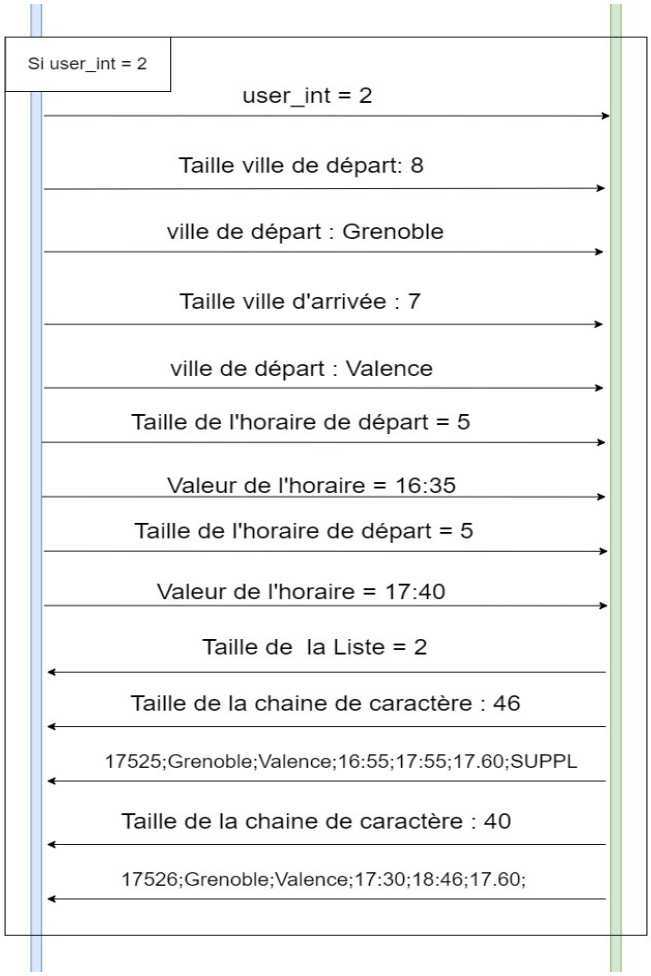
Avec l'appel de la fonction `sendTrainbyHourAndCity` le serveur quant à lui commence par récupérer les données transmises par le client, la ville de départ, la ville d'arrivée et l'horaire de départ.

Après la récupération des données, on invoque une autre fonction, il s'agit du controlleur `getTrainByGivenDepartureCity` qui va recevoir en paramètre les données transmises par le client et le fichier sncf.txt . Ce dernier va utiliser la fonction `getTrainBy_Departure_AND_Arrival` qui va renvoyer un tableau des lignes qui nous intéressent par rapport aux villes de départ et villes d'arrivée. Ensuite la fonction va réutiliser ce tableau pour maintenant comparer les horaire des lignes et l'horaire passé en paramètre, car si l'horaire est bon, la fonction va renvoyer la bonne ligne et si il ne correspond pas le train choisi sera le train qui à l'horaire le plus proche supérieur et si il n'y en à pas la fonction retourne le premier train qui fait ce trajet (le lendemain).

La chaîne de caractère que cette fonction aura retourné va être envoyée au client. Ensuite le client va d'abord lire la taille de cette chaîne, puis il va lire la valeur de la chaîne de caractère pour pouvoir l'afficher sur la sortie standard.

☐ Consultation des trains correspondant à une ville de départ, à une ville d'arrivée ,à un horaire de départ et à une tranche horaire.

Pour consulter la liste des trains de la journée correspondant à une ville de départ, à une ville d'arrivée et à une tranche horaire, il faut choisir l'option 2 dans le menu principal. Cela invoque la fonction getAllTrainsWithGivenSlotTime qui prend une socket de service en paramètre et qui transmet l'option choisie au serveur ainsi que les données de l'utilisateur à travers les fonction auxiliaires askAndSendDeparture,askAndSendArrival et askAndSendTime qui interagissent avec l'utilisateur pour récuperer ses données et les envoyer au serveur.



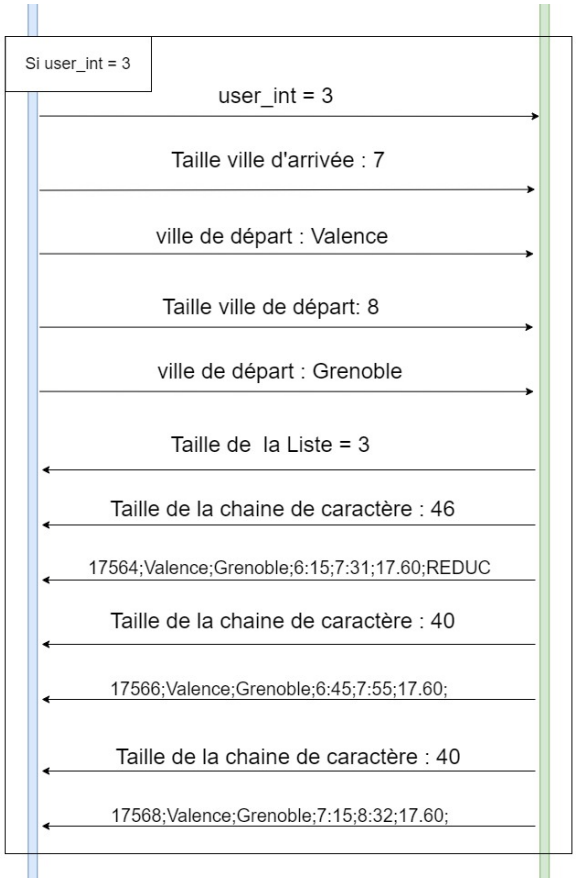
Avec l'appel à la fonction `sendTrainsOverSlotTime` le serveur quant à lui commence par récupérer les données transmises à savoir la ville de départ, la ville d'arrivée et la tranche horaire composée de deux bornes.

Une autre fonction est invoquée après la récupération des données client, il s'agit du controlleur getTrainsOverSlotTime . Ce dernier effectue la recherche dans le fichier sncf.txt et construit la liste des trains demandée qui est envoyée au client via la fonction sendListeTrains.

Le client à sa réception de la réponse, lit d'abord la longueur de la liste et boucle autant de fois pour lire la longueur du train pour pouvoir lire par la suite la chaîne exacte correspondant aux données du train et afficher chaque train sur la sortie standard.

☐ Consultation des trains correspondant à une ville de départ et à une ville d'arrivée avec le meilleur prix ou ayant le trajet optimum.

Pour consulter la liste des trains de la journée correspondant à une ville de départ et à une ville d'arrivée avec le meilleur prix, il faut choisir l'option 3 dans le menu principal.

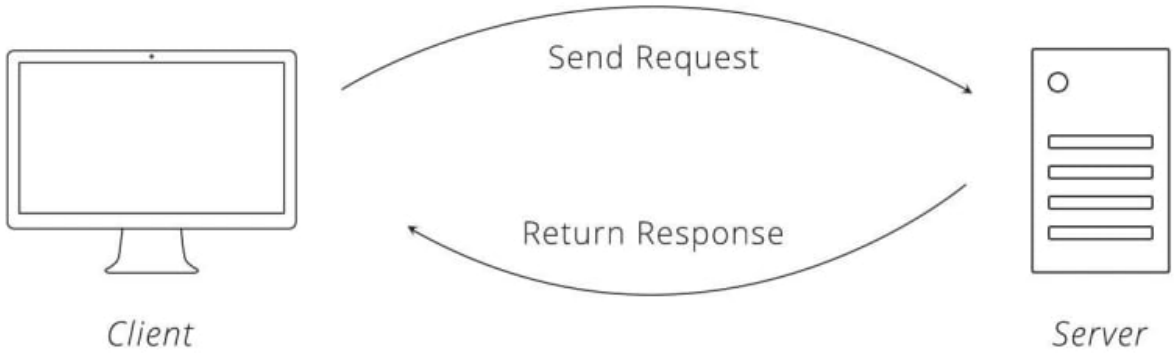


Cela invoque la fonction getServeurTrainBy_Departure_AND_Arrival qui prend une socket de service en paramètre et qui transmet l'option choisie au serveur ainsi que les données de l'utilisateur (ville de départ et ville d'arrivée) à l'aide des fonction auxiliaires `askAndSendDeparture` **`et`** `askAndSendArrival.`

La différence de cette fonctionnalité avec les autres est que le traitement demandé c'est à dire l'application des formules de calculs des nouveaux prix aux cas d'une réduction (REDUC) ou d'une hausse de prix (SUPPL) est faite dans le client-side.

Le serveur récupère la liste des trains en invoquant la fonction getTrainBy_Departure_AND_Arrival qui ouvre le fichier sncf.txt et effectue la recherche lisant ligne par ligne et comparant la ville de départ et d'arrivée de chaque train courant avec celles en paramètres. Cette dernière renvoie la liste finale de trains qui est envoyée au client avec la fonction sendListeTrains.

Le client à sa réception de la réponse, lit d'abord la longueur de la liste et boucle autant de fois pour lire la longueur du train pour pouvoir lire par la suite la chaîne exacte correspondant aux données du train et afficher chaque train sur la sortie standard mais avant l'affichage finale les flags RÉDUC et SUPPL sont vérifiés, autrement dit s'ils existent une réduction de 20% est appliquée au prix initial et une hausse de prix de 10% au cas de SUPPL serait également appliquée.



Pour résumer, chaque interrogation du client est associée à un handler pour la gérer. Toutes les actions possibles de l'utilisateur côté client sont surveillées côté serveur.

7- Expérimentation et exemple d'exécution

Pour commencer nous avons besoin de compiler. Pour cela nous avons créé un Makefile qui crée deux exécutables, **mainClient** et **mainServeur**. Dans un premier temps il est question de compiler les fichiers et lancer le serveur puis ensuite lancer le client. Sur la machine serveur, il suffit de taper les commandes suivantes :

>>make

```
root@Pavilion-gaming-laptop:/mnt/c/Users/loicd/Desktop/SetR/sources# make
gcc -g -Wall -c  mainClient.c
gcc -g -Wall -c  client.c
gcc -o mainClient -lm  mainClient.o client.o
gcc -g -Wall -c  mainServeur.c
gcc -g -Wall -c  serveur.c
gcc -g -Wall -c  sncf.c
gcc -o mainServeur -lm  mainServeur.o serveur.o sncf.o
```

>>./mainServeur Port ../doc/sncf.txt

```
root@Pavilion-gaming-laptop:/mnt/c/Users/loicd/Desktop/SetR/sources# ./mainServeur
 8082 /mnt/c/Users/loicd/Desktop/SetR/doc/sncf.txt
Server successfully configured!
App runs on port 8082
Waiting for connection ...
```

Avec « Port » qui est le numéro de port TCP choisi pour le passage des données, « ../doc/sncf.txt » est le chemin absolu du fichier texte contenant la base de données (ici sncf.txt).

Arrivé à cette étape nous avons lancé le serveur. Il faut maintenant lancer client, pour cela nous avons besoin de taper les commandes suivantes :

>>./mainClient hostname Port

```
root@Pavilion-gaming-laptop:/mnt/c/Users/loicd/Desktop/SetR/sources# ./mainClient localhost 8082
```

Avec « hostname » qui est l'adresse de la machine serveur et « Port » le port TCP choisi pour le passage des données.

Maintenant que nous avons exécuté les fichiers et initialisé les machines avec leurs sockets, le terminal nous affiche le menu principal. Plusieurs choix s'offrent à nous (voir page 6 de ce document).

```
Successfully connected to the server!

**********************************************************************************************************
**********************************    MAIN MENU    ********************************************************
**********************************************************************************************************

Choose an option :
    0 : See all trains
    1 : Get Train By given departure city,arrival city and departure time
    2 : Get Train By given departure city,arrival city and departure slot time
    3 : Get Train By given departure and arrival city with best prices or optimal journey time
   -1 : Quit

**********************************************************************************************************
**********************************************************************************************************
**********************************************************************************************************
option >>:
```

Choix n°0 :

Si l'on choisit l'option 0 le serveur traite votre demande et nous renvoie l'entièreté de la base de données. Il faut ensuite taper q pour revenir au menu principal.

```
option >>: 0
>>> HERE IS THE LIST OF FOUND TRAINS  <<<

               Train N° 17564  Valence -> Grenoble    6:15    7:31    -  14.08 $

               Train N° 17566  Valence -> Grenoble    6:45    7:55    -  17.60 $

               Train N° 17568  Valence -> Grenoble    7:15    8:32    -  17.60 $

               Train N° 17524  Grenoble -> Valence    16:30    17:45    -  17.60 $

               Train N° 17525  Grenoble -> Valence    16:55    17:55    -  19.36 $

               Train N° 17526  Grenoble -> Valence    17:30    18:46    -  17.60 $

               Train N° 17528  Grenoble -> Valence    18:30    19:45    -  14.08 $

               Train N° 86181  Valence -> Montelimar    12:30    12:56    -  7.84 $

               Train N° 86183  Valence -> Montelimar    14:10    14:30    -  10.78 $

               Train N° 86187  Valence -> Montelimar    16:30    16:56    -  9.80 $

               Train N° 86174  Montelimar -> Valence    6:30    6:55    -  9.80 $

               Train N° 86175  Montelimar -> Valence    7:10    7:30    -  10.78 $

               Train N° 86177  Montelimar -> Valence    7:30    7:55    -  9.80 $

               Train N° 86178  Montelimar -> Valence    8:00    8:25    -  9.80 $

               Train N° 86179  Montelimar -> Valence    8:30    8:55    -  7.84 $

               Train N° 9862  Valence -> Paris Gare de Lyon    15:15    17:49    -  87.60 $

               Train N° 6194  Valence -> Paris Gare de Lyon    16:15    19:15    -  92.00 $

               Train N° 6208  Valence -> Paris Gare de Lyon    17:15    19:53    -  92.80 $

               Train N° 6035  Paris Gare de Lyon -> Valence    7:41    10:11    -  90.40 $

               Train N° 9713  Paris Gare de Lyon -> Valence    10:07    12:19    -  90.00 $

               Train N° 6063  Paris Gare de Lyon -> Valence    10:07    13:15    -  92.20 $

press q to quit :
```

Choix n°1 :

Si l'on choisit l'option 1, le client envoie la requête au serveur et le serveur attend les arguments de la requête. Le client demande alors à l'utilisateur le nom de la ville de départ du train, demande le nom de la ville d'arrivée puis l'horaire voulu par l'utilisateur pour le trajet.

```
option >>: 1
############## DEPARTURE ##########
Enter Departure Town Or Train Station :
paris gare de lyon
############## ARRIVAL ############
Enter Arrival Town Or Train Station :
ValENCE
############## TIME DEPARTURE ############
Enter Time limit 1 :  [ Requested format HH:mm ] : 12:45
```

Le résultat rendu par le serveur est la ligne de train demandée par l'utilisateur avec le prochain horaire de départ.

```
>>> THE NEXT TRAIN GOING FROM  Paris Gare de Lyon  to  Valence :

        Train N° 6035  Paris Gare de Lyon -> Valence    7:41    10:11    -  90.40 $

press q to quit :
```

Choix n°2:

Dans le cas de l'option 2, le programme client demande à l'utilisateur de rentrer une ville de départ, une ville d'arrivée, un horaire de départ minimum et un horaire de départ maximum.

```
option >>: 2
############## DEPARTURE ##########
Enter Departure Town Or Train Station :
grenoble
############## ARRIVAL ############
Enter Arrival Town Or Train Station :
valence
############## TIME DEPARTURE ############
Enter Time limit 1 :  [ Requested format HH:mm ] : 12:33
############## TIME DEPARTURE ############
Enter Time limit 2 :  [ Requested format HH:mm ] : 20:45
## 17524;Grenoble;Valence;16:30;17:45;17.60;
## 17525;Grenoble;Valence;16:55;17:55;17.60;SUPPL
## 17526;Grenoble;Valence;17:30;18:46;17.60;
## 17528;Grenoble;Valence;18:30;19:45;17.60;REDUC
```

Le serveur reçoit la demande et renvoie les trajets correspondants à la recherche de l'utilisateur. De plus, le client affiche le train ayant le tarif le plus abordable et le trajet qui prend le moins de temps.

```
>>> HERE ARE THE TRAINS GOING FROM Grenoble To Valence <<< (found 4 )

                Train N° 17524  Grenoble -> Valence    16:30    17:45    -  17.60 $

                Train N° 17525  Grenoble -> Valence    16:55    17:55    -  19.36 $

                Train N° 17526  Grenoble -> Valence    17:30    18:46    -  17.60 $

                Train N° 17528  Grenoble -> Valence    18:30    19:45    -  14.08 $
----------------------
The lower price for this travel is 14.08 for
Train N°: 17528 leaving at 18:30 and arriving at 19:45
----------------------
----------------------
The fastest train for this travel is
Train N°: 17525 leaving at 16:55 and arriving at 17:55 for 60 minutes of travel
    price : 17.60 $
----------------------

press q to quit :
```

Choix n°3 :

Le client demande à l'utilisateur de rentrer la ville de départ et la ville d'arrivée. Le serveur traite la demande et renvoie la liste de trajet.

```
option >>: 3
############## DEPARTURE ##########
Enter Departure Town Or Train Station :
paris gare de lyon
############## ARRIVAL ############
Enter Arrival Town Or Train Station :
valence
```

Le client affiche la liste renvoyée par le serveur puis affiche le trajet le plus intéressant en termes de prix et le trajet le moins cher disponible pour les villes de départ et d'arrivée choisies.

```
## 6035;Paris Gare de Lyon;Valence;7:41;10:11;113.00;REDUC
## 9713;Paris Gare de Lyon;Valence;10:07;12:19;90.00;
## 6063;Paris Gare de Lyon;Valence;10:07;13:15;92.20;
>>> HERE ARE THE TRAINS GOING FROM Paris Gare de Lyon To Valence <<< (found 3 )

                Train N° 6035  Paris Gare de Lyon -> Valence    7:41    10:11    -  90.40 $

                Train N° 9713  Paris Gare de Lyon -> Valence    10:07    12:19    -  90.00 $

                Train N° 6063  Paris Gare de Lyon -> Valence    10:07    13:15    -  92.20 $
----------------------
The lower price for this travel is 90.00 for
Train N°: 9713 leaving at 10:07 and arriving at 12:19
----------------------
----------------------
The fastest train for this travel is
Train N°: 9713 leaving at 10:07 and arriving at 12:19 for 132 minutes of travel
    price : 90.00 $
----------------------

press q to quit :
```

Choix n°-1 :

Le client envoie -1 au serveur. A la réception, le serveur tue le fils ce qui coupe la connexion.

```
option >>: -1
root@Pavilion-gaming-laptop:/mnt/c/Users/loicd/Desktop/SetR/sources#
```

Make clean :

Le makefile possède une cible clean qui permet de supprimer tous les fichiers objets créés par les compilations précédentes.

```
root@Pavilion-gaming-laptop:/mnt/c/Users/loicd/Desktop/SetR/sources# make clean
rm -rf *.o mainClient mainServeur
```

8- Conclusion

Ce projet nous a permis d'avoir une expérience concrète dans le domaine de la programmation réseau et système en C et d'affiner nos connaissances sur les sockets, le protocole TCP/IP et la gestion des processus. La réalisation d'une application fonctionnelle répondant aux besoins spécifiques de consultation des horaires de trains a abouti à un projet complet et bien structuré.

En effet la conception logicielle globale du projet a été menée de manière méthodique, depuis la définition des objectifs jusqu'à la phase de mise en production. Cette manière de travailler pour le projet nous à appris une autre façon de concevoir et mener un projet.

La conception modulaire du projet, adoptant une structure similaire au pattern Model-View-Controller, a permis une organisation claire et une gestion efficace des différentes fonctionnalités.

Nous avons ajouté des fonctionnalités au projet comme :

- la gestion de la mort du processus père lors d'un ^C (simulation d'un potentiel crash ou fermeture du serveur) côté serveur;
- la fonctionnalité d'affichage de toute la base de donnée (option 0);
- la gestion de la mort du fils suite à une fin de session de la part du client (option -1);
- la gestion de la mort du processus fils lors d'un ^C (simulation d'un potentiel crash) côté client.