Année 2023/2024

Université Grenoble Alpes
L3 MIAGE

Projet Système/Réseaux Application client/serveur TCP/IP

Réalisé par :

EL-BOUCH Ismail

MAHRI Yassine

DING ZICHEN

XIAOMAN Liu

Université Grenoble Alpes
L3 MIAGE

1.Introduction:

Objectif du projet ;

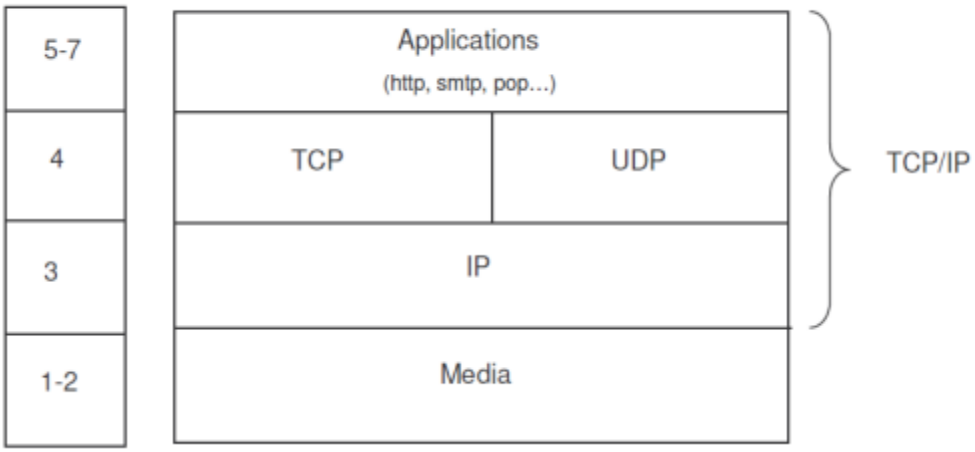
Le projet consiste à développer en langage C une application client/serveur TCP/IP simplifiée permettant aux utilisateurs de consulter des listes de trains disponibles entre deux villes données. L'objectif est de fournir des fonctionnalités telles que la recherche de trains en fonction de critères tels que la ville de départ, la ville d'arrivée, l'horaire de départ, etc. Les informations sur les trains sont stockées dans un fichier texte au format spécifié.

2.Contexte Général:

Au cours de notre projet, nous avons employé des concepts liés aux réseaux, que nous exposerons initialement dans cette section.

2.1.Protocole TCP/IP

TCP/IP représente une suite de protocoles essentielle pour la connectivité des réseaux à l'échelle planétaire. Cette famille de protocoles joue un rôle crucial en permettant la communication et l'échange d'informations entre diverses infrastructures réseau, favorisant ainsi la connectivité mondiale



La couche réseau (couche 3) est mise en œuvre par le protocole IP, responsable de la gestion de l'acheminement des paquets vers les couches supérieures sans établir de connexion, sans garantir d'ordre ni de bon acheminement. L'identification de la machine cible se fait à travers son adresse IP.
Quant à la couche transport (couche 4), elle peut être gérée par deux protocoles distincts : TCP et UDP. Ces protocoles définissent les caractéristiques de la communication, influençant ainsi le mode de transfert des données.

TCP assure une communication en mode connecté, offrant fiabilité grâce aux acquittements, contrôle d'erreurs, contrôle de flux, et garantit l'ordre des paquets. En revanche, UDP propose une communication en mode non connecté, sans contrôle de flux ni acquittement, et ne garantit pas l'ordre des messages.

Schéma des échanges pour la mise en place d'une connexion TCP :

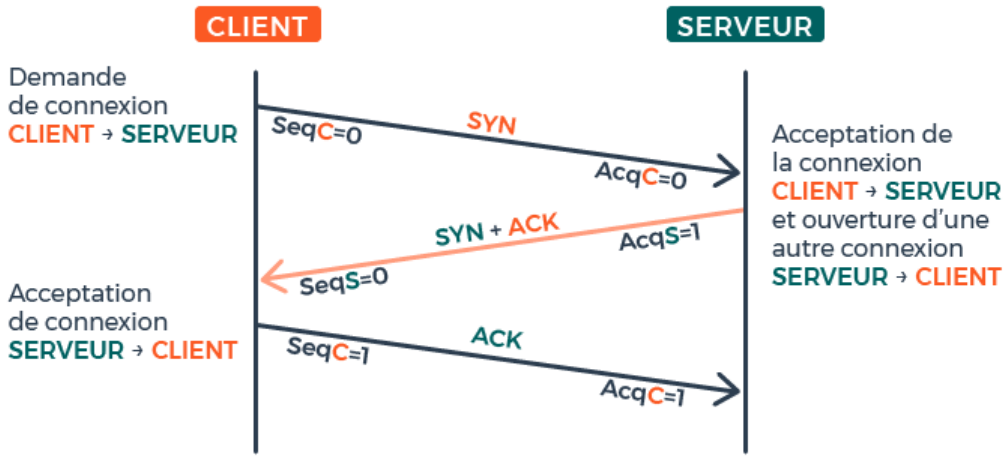
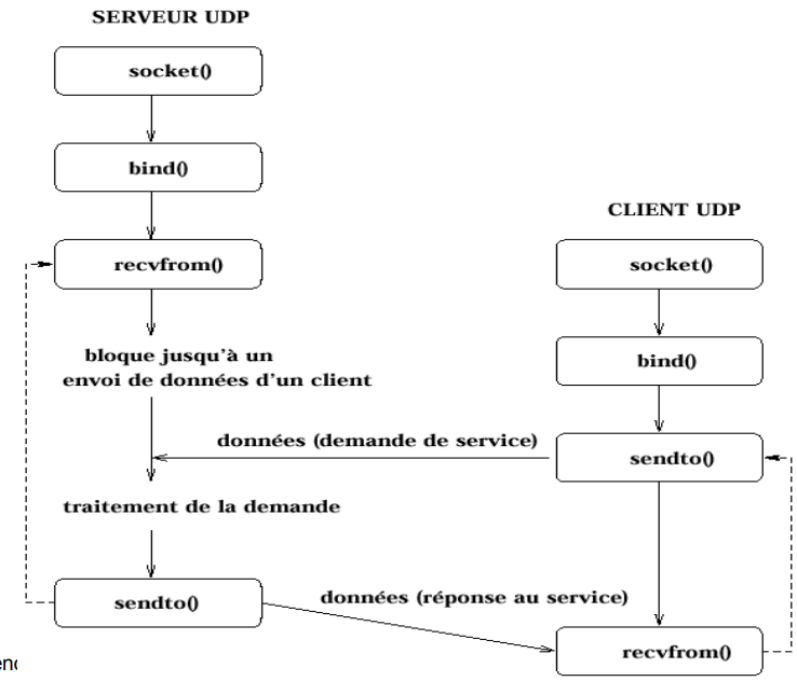
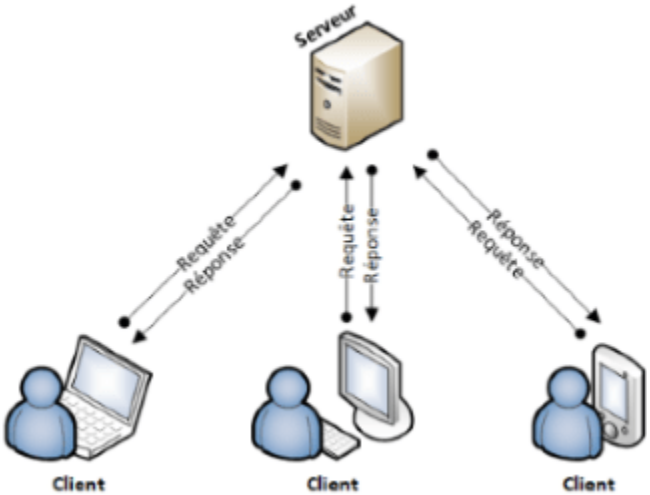


Schéma de communication socket UDP:



dans notre projet on va utiliser le protocole TCP/IP.

2.2.Architecture réseau :



L'architecture client-serveur constitue le contexte dans lequel des applications sur des machines clientes interagissent avec des applications sur des machines serveurs. Un exemple courant est le navigateur Web d'un client, sollicitant (par le biais d'une "requête") le contenu d'une page Web à un serveur Web qui lui renvoie le résultat (appelé "réponse").

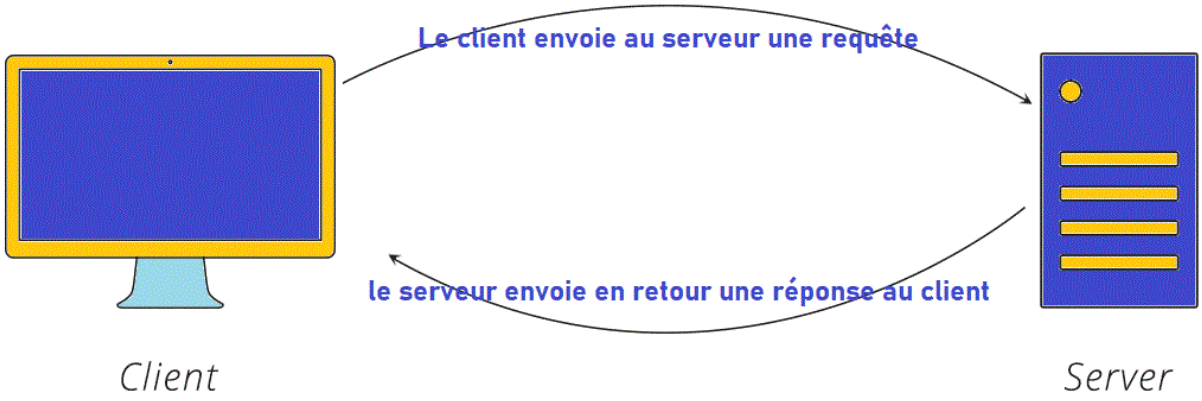
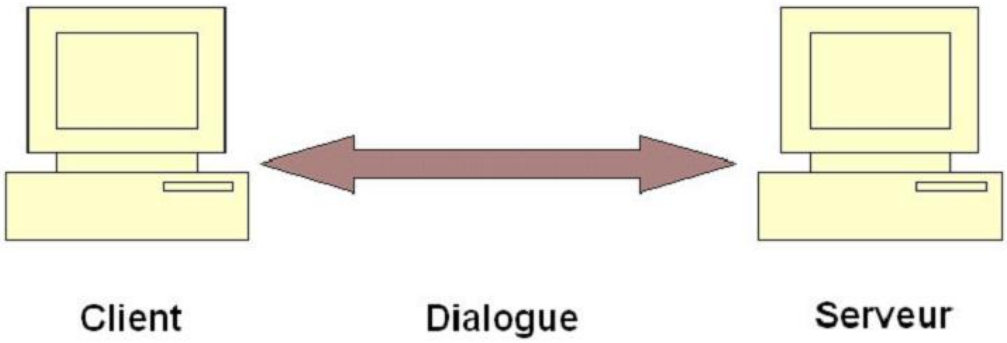
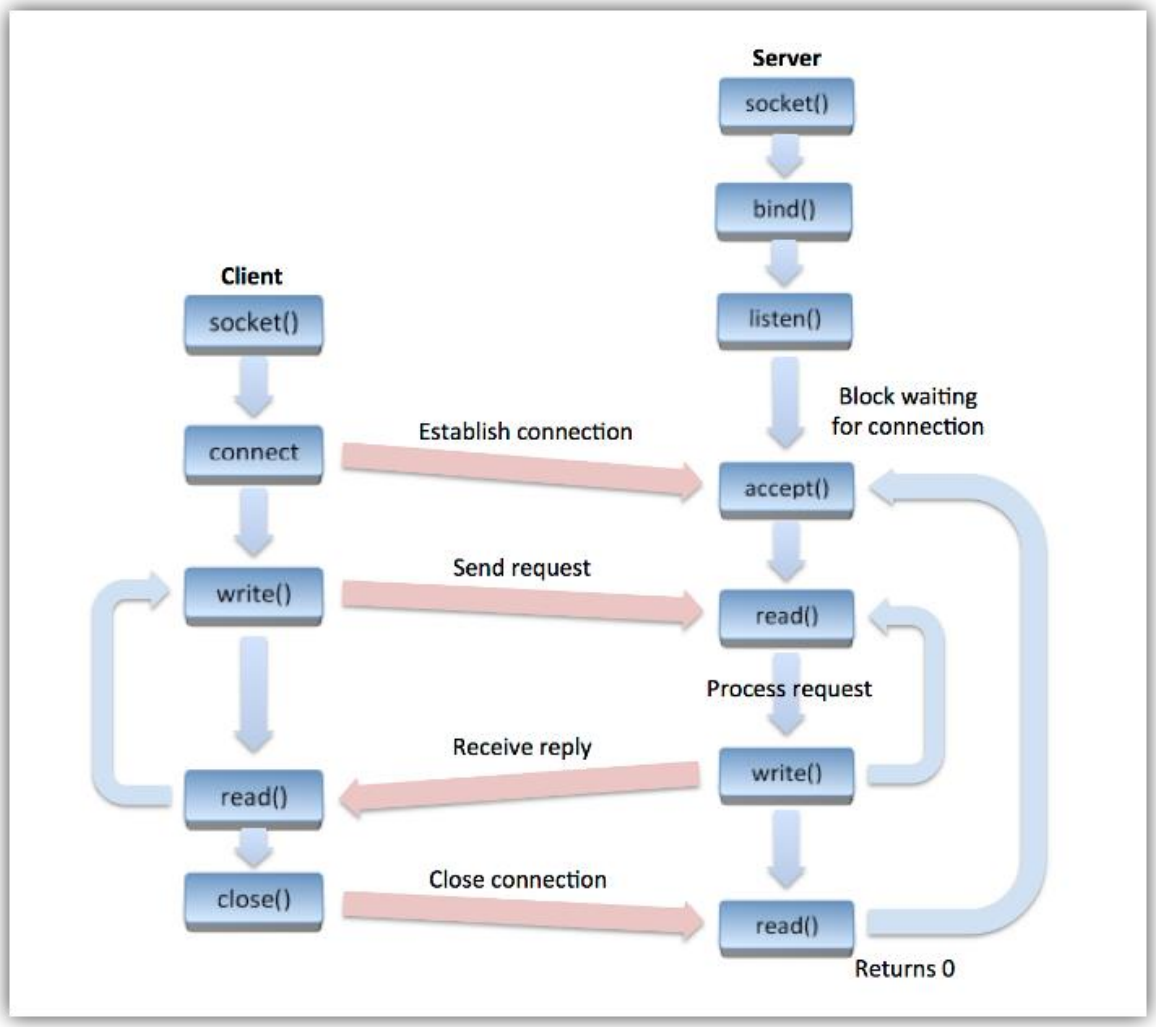


Schéma utilisé dans le projet :



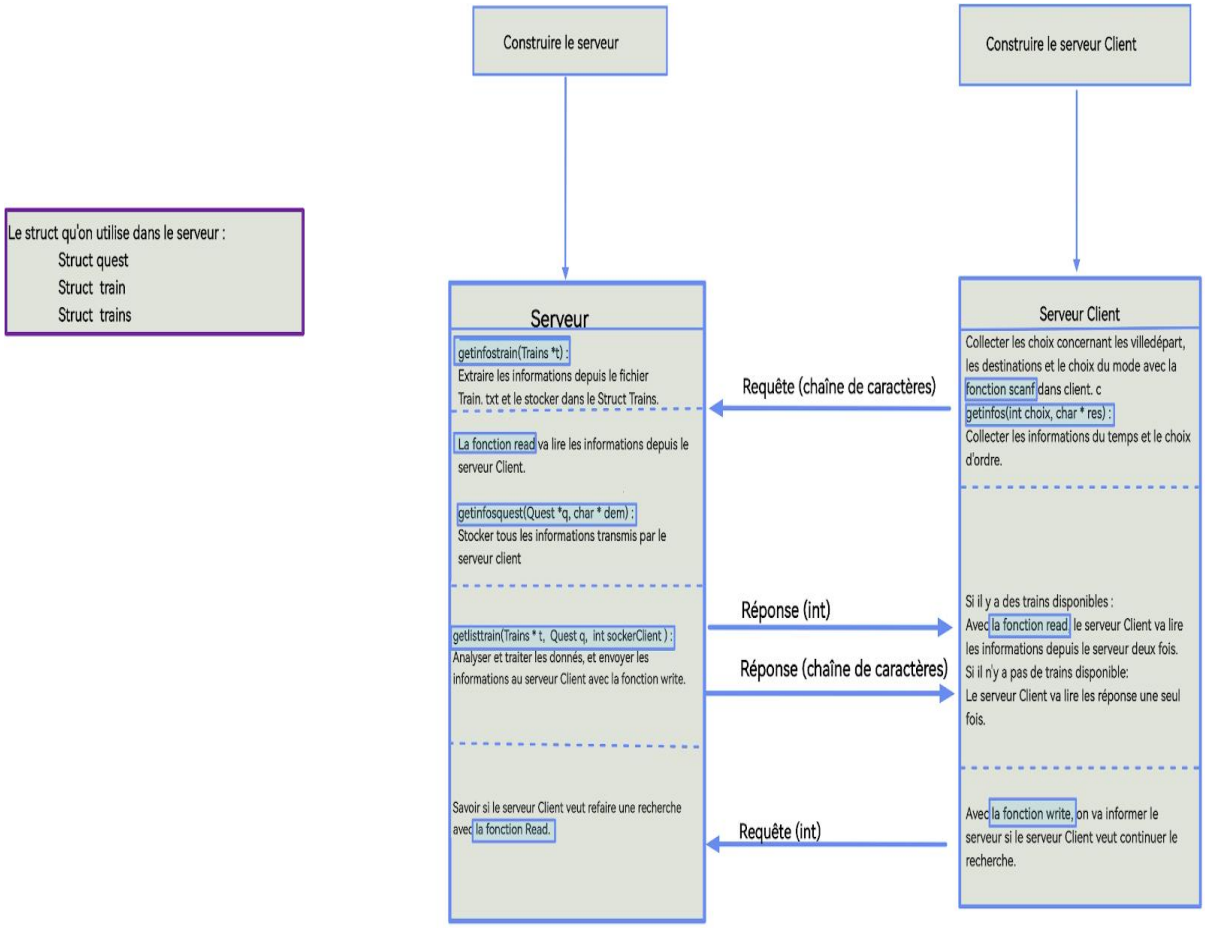
Le client est le programme qui prend l'initiative d'établir le contact avec le serveur et d'envoyer des requêtes, assurant ainsi toujours le rôle de celui qui démarre le dialogue.

mode connecté TCP :



3.Analyse et fonctionnalités de l'application:

L'application met à la disposition des utilisateurs du service les fonctionnalités suivantes :



Présentation de la conception logicielle :

Le programme est un système client-serveur permettant de rechercher des informations sur les trains en fonction de différents critères tels que la ville de départ, la ville d'arrivée, le temps de départ... Le système est constitué de deux parties principales : le client (client.c) et le serveur (fonc_ser.c, main.c).

Le client.c est responsable de la communication avec l'utilisateur. Il prend en entrée les informations nécessaires pour la recherche de trains, établit une connexion avec le serveur, envoie les requêtes au serveur et affiche les résultats. En effet on a défini 3 mode :
mode 1 : avec un temps précisé, en spécifiant une plage horaire définie, avec indication de l'heure de départ, qui identifie de manière précise un unique train, le plus proche en réponse à la requête spécifiée (ville de départ, ville d'arrivée, heure de départ).
mode 2 : avec une période de temps, en précisant une période de temps déterminée, avec indication de l'heure de départ et de l'heure d'arrivée, qui identifie plusieurs trains dans la plage horaire spécifiée.

mode 3 : sans indiquer le temps; sans spécifier l'heure, affichant directement tous les trains entre deux villes sans demander d'entrer l'heure de départ et d'arrivée.

Le serveur est composé de deux fichiers, fonc_ser.c et fonc_ser.h, qui contiennent les fonctions nécessaires pour gérer les requêtes des clients. Le serveur lit les informations sur les trains à partir du fichier "train.txt", puis traite les demandes des clients et renvoie les résultats.
-main.c : Cette partie du programme a pour rôle de construire le serveur et d'établir la communication entre le serveur (main.c) et le client (client.c). Le fichier main.c contient la fonction principale qui initialise le serveur en spécifiant le numéro de port sur lequel le serveur écoutera les connexions entrantes.La séquence d'exécution commence par la création du socket serveur à l'aide de la fonction socket. Ensuite, la fonction bind est utilisée pour lier le socket à une adresse IP et un numéro de port spécifiques. La fonction listen permet au serveur d'attendre les connexions des clients.Une fois que le serveur est configuré et en attente de connexions, il entre dans une boucle infinie, utilisant la fonction accept pour accepter les connexions entrantes des clients. Pour chaque nouvelle connexion, le serveur crée un processus enfant (fonction fils) à l'aide de la fonction fork, ce qui permet une gestion simultanée des requêtes de plusieurs clients.

-fonc_ser.c : La première étape de cette partie du programme consiste à créer un processus fils à l'aide de la fonction fork. Ce processus fils sera chargé d'analyser toutes les données reçues du client.À l'intérieur de cette boucle, la fonction getinfostrain(t) est employée pour extraire les informations du fichier "train.txt". Ces données sont ensuite stockées dans la structure train (déclarée dans fonc_ser.h), facilitant leur manipulation ultérieure.Ensuite, le serveur lit les informations provenant du client en utilisant la fonction read. De plus, la fonction getinfoquest est utilisée pour stocker les informations du client dans la structure quest (déclarée dans fonc_cli.h). Cela permet d'organiser les données de manière structurée pour une analyse plus efficace.La fonction getlisttrain est ensuite employée pour comparer les données reçues du client avec celles extraites du fichier "train.txt". Cette comparaison permet d'obtenir les résultats recherchés, qui sont ensuite renvoyés au client par le biais de la communication socket.

Explication détaillée du protocole d'échange :

Le protocole d'échange entre le client et le serveur est basé sur la communication par sockets. Le serveur utilise la fonction accept pour accepter les connexions des clients. Une fois la connexion établie, le serveur crée un processus enfant (fonction fils) pour gérer la demande du client. Les informations sur les trains sont stockées dans un fichier texte ("train.txt") et sont lues par le serveur pour traiter les requêtes.

Le client envoie une requête au serveur sous la forme d'une chaîne de caractères structurée(dem), contenant des informations telles que le mode de recherche, les villes de départ et d'arrivée, le temps. En stockant toutes les informations temporelles avec la fonction “getinfos”, nous utilisons ensuite la fonction “write” pour transmettre toutes ces informations au serveur . Le serveur utilise la fonction “read” pour recevoir la requête du client, puis la traite en fonction du mode de recherche spécifié.

Après traitement, le serveur envoie les résultats au client avec une chaîne de caractères structurée.Il y a deux possibilités pour la réception des données. Le serveur envoie deux réponses au client. La première concerne le nombre de trains qui répondent aux critères, utilisant un entier comme support. La deuxième concerne les informations sur les trains, utilisant une chaîne de caractères comme support. Dans **client.c**, nous commençons par vérifier si le nombre de trains renvoyé lors de la première réponse est égal à zéro. Si c'est le cas, nous afficherons directement "pas de train disponible". Sinon, nous lisons la deuxième réponse.
La communication entre le client et le serveur est également gérée à l'aide des fonctions write et read pour envoyer et recevoir des données sur la connexion.

Connexion :

- Le serveur attend des connexions entrantes sur un certain port.
- Le client se connecte au serveur en spécifiant l'adresse IP et le port.

Dans le fichier **main.c (serveur)** :

Création du Socket Serveur :

```
socketServer = socket(AF_INET, SOCK_STREAM, 0)
```

AF_INET spécifie la famille d'adresses (IPv4 dans ce cas).
SOCK_STREAM indique que le socket sera utilisé pour une communication orientée connexion (TCP).

Initialisation de l'adresse du serveur :

```
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(port);
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

htons et htonl effectuent la conversion de l'ordre des octets pour le format réseau.

Liaison du Socket à une Adresse :

```
bind(socketServer, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
```

Associe le socket à une adresse IP et un numéro de port.

Écoute pour les Connexions Entrantes :

```
listen(socketServer, 5);
```

Met le socket en mode écoute, prêt à accepter les connexions entrantes.

Acceptation d'une Connexion :

sockerClient = accept(socketServer, (struct sockaddr*)&clientAddr, &sizead);
Attend qu'un client se connecte et accepte la connexion. sockerClient est un nouveau socket dédié à la communication avec ce client particulier.

Échange de données :

- Une fois connecté, le client et le serveur communiquent en utilisant des sockets TCP/IP.
- Les données sont échangées sous forme de chaînes de caractères (e.g., char *).
- Le client envoie des requêtes au serveur, et le serveur répond avec les résultats.

Réception de la demande du client :

Le serveur attend la réception d'une chaîne de caractères envoyés par le client, qui représente la demande du client. La demande est envoyée par le client en utilisant la fonction **write** dans le fichier **client.c** et reçue côté serveur avec la fonction **read** dans **fonc_ser.c.**

```
char *dem=(char *)malloc(sizeof(char)*100);
read(sockerClient, dem, sizeof(char)*100);
```

Le contenu de cette chaîne est ensuite utilisé pour initialiser une structure Quest qui représente la demande du client.

```
Quest demande;
iniquest(&demande);
getinfosquest(&demande, dem);
```

Traitement de la demande :

La demande du client est ensuite utilisée pour effectuer différentes opérations côté serveur, telles que la recherche d'informations sur les trains en fonction des critères spécifiés par le client.

```
getlisttrain(t, demande, sockerClient);
```

Envoi des résultats au Client :

Les résultats de la recherche ou d'autres informations pertinentes sont ensuite envoyés au client. La fonction getlisttrain remplit un tableau de trains (s'il existe des trains disponibles) répondant aux critères spécifiés et envoie ces informations au client.

```
write(sockerClient,&lentp,sizeof(int));
write(sockerClient, res, sizeof(char)*2000);
```

Réception de la confirmation de fin de recherche du client :

Enfin, le serveur attend de recevoir une confirmation du client indiquant s'il souhaite terminer la recherche ou non.

```
read(sockerClient, &end, sizeof(int));
```

Le serveur utilise cette information pour déterminer s'il doit continuer à traiter les demandes du client ou s'il doit terminer la connexion avec le client.

Requêtes du client :

- Le client envoie des requêtes au serveur pour obtenir des informations sur les trains.
- Les requêtes sont construites en incluant des informations telles que la ville de départ, la ville d'arrivée, le mode de recherche (par temps, par période, sans indication de temps), etc.
- Les requêtes peuvent également inclure des informations sur le temps de départ plus tôt ou tard selon le mode sélectionné.

Dans le fichier client.c :

Création de la Requête :

```
// Collecter les informations nécessaires pour la requête
char choix='\0';
…

// Création de la requête sous forme de chaîne de caractères
char *villedepart=(char *)malloc(sizeof(char)*100);
char *villearrive=(char *)malloc(sizeof(char)*100);
char *dem=(char *)malloc(sizeof(char)*100);
memset(dem,0,sizeof(dem));

// ... (saisie des informations nécessaires)

sprintf(dem+strlen(dem),"%c",choix);
strcat(dem,";");
strcat(dem,villedepart);
strcat(dem,";");
strcat(dem,villearrive);
strcat(dem,";");
```

3.2. Envoi de la Requête au Serveur (client.c)

La requête est envoyée au serveur à l'aide de la fonction write.

```
write(sockerClient, dem, sizeof(char)*100);
```

3.3. Réception de la Requête du Client (fonc_ser.c)

La réception de la requête côté serveur est gérée dans la fonction fils du fichier fonc_ser.c.

```
char *dem=(char *)malloc(sizeof(char)*100);
read(sockerClient, dem, sizeof(char)*100);

Quest demande;
iniquest(&demande);

// Extraction des informations de la requête
getinfosquest(&demande, dem);
```

- 3.4. Traitement de la Requête (fonc_ser.c)

Le traitement de la requête, tel que la récupération des informations des trains et la préparation des résultats, est effectué dans la fonction fils du fichier fonc_ser.c.

```
Trains *t;
t=(Trains *)malloc(sizeof(Trains)*100);
t->len=0;

// Récupération des informations des trains
getinfostrain(t);
// Préparation des résultas
getlisttrain(t,demande,sockerClient);
```

C'est dans cette fonction que le serveur peut effectuer des opérations en fonction des informations reçues dans la requête du client. La gestion des trains et la préparation des résultats sont incluses dans cette partie du code.

Traitement des requêtes côté serveur :

- Le serveur reçoit les requêtes du client.
- Il traite ces requêtes en interagissant avec un ensemble de données stockées dans un fichier texte (train.txt).
- Le serveur utilise ces données pour rechercher des informations sur les trains qui correspondent aux critères spécifiés dans la requête.

Réponses du serveur :

- Le serveur renvoie les résultats au client.
- Les résultats sont les trains qui répondent aux critères spécifiés dans la requête en incluant des détails tels que le numéro de train, la ville de départ, les horaires de départ et tranche horaire, le prix, etc.

- Les réponses peuvent être formatées sous forme de chaînes de caractères(s'il existe des trains disponibles) et entier(int)(le nombre de trains disponibles).

Pour gérer les réponses du serveur, on a se servi de la fonction getlisttrain de fonc_ser.c. Cette fonction traite les informations concernant les trains disponibles et construit la réponse qui sera renvoyée au client.
Cette fonction utilise la fonction getResultat pour coller les informations séparées pour chaque train. Ensuite, en fonction du mode de recherche spécifié dans la requête du client (q.modec), la fonction envoie les résultats au client via la connexion socket.

Fin de la communication :

- Le client peut choisir de terminer la communication en indiquant au serveur qu'il souhaite mettre fin à la recherche.
- Le serveur gère cette demande et peut terminer la connexion ou attendre de nouvelles requêtes du client.

Conception et mise en œuvre des fonctionnalités :

Les fonctionnalités du programme incluent la recherche de trains en fonction de différents critères tels que la ville de départ, la ville d'arrivée, le temps, etc. Le serveur utilise des structures de données pour stocker les informations sur les trains et utilise des fonctions pour traiter les demandes des clients.

Les fonctions getinfostrain, getinfosquest, getlisttrain, compareTime, et getResultat sont responsables de divers aspects du traitement des demandes des clients, notamment la récupération des informations sur les trains, l'analyse des requêtes, la recherche de trains correspondants, la comparaison des temps, et la création des résultats à renvoyer au client.

Exemples d'expérimentations :

Pour expérimenter avec le programme, vous pouvez exécuter le serveur en utilisant la commande ./main <port> (où <port> est le numéro de port sur lequel le serveur écoutera). Ensuite, vous pouvez exécuter le client en utilisant la commande ./client <port> <adresse du serveur>(en spécifiant le même numéro de port et adresse du serveur est optionnelle ,par défaut <localhost>). Le client vous guidera à travers les différentes étapes pour effectuer une recherche de trains en fonction de vos critères.

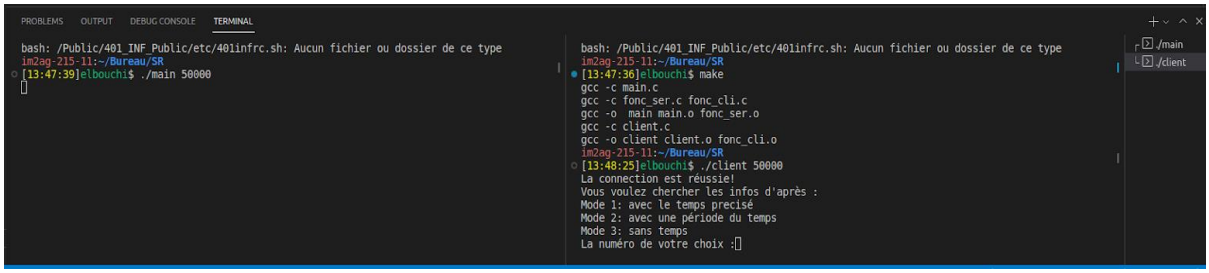
Conclusion sur le travail effectué :

Le programme semble être conçu pour répondre aux besoins de recherche d'informations sur les trains. Cependant, il est important de noter que le code peut nécessiter des améliorations, notamment en termes de gestion d'erreurs, de sécurité et de convivialité de l'interface utilisateur. Une expansion future du programme pourrait inclure des fonctionnalités supplémentaires, une gestion plus robuste des erreurs, et éventuellement une interface utilisateur graphique pour améliorer l'expérience globale.

Université Grenoble Alpes
L3 MIAGE

4.tests:

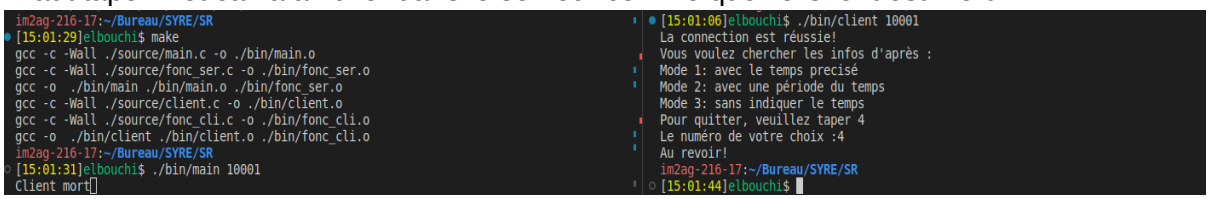
compilation et exécution:



Pour compiler on utilise make, et pour exécuter on commence d'abord par ./main <numéro de port>, puis le serveur attend la connexion de client avec le même numéro de port ./client <même numéro de port>

Test pour quitter:

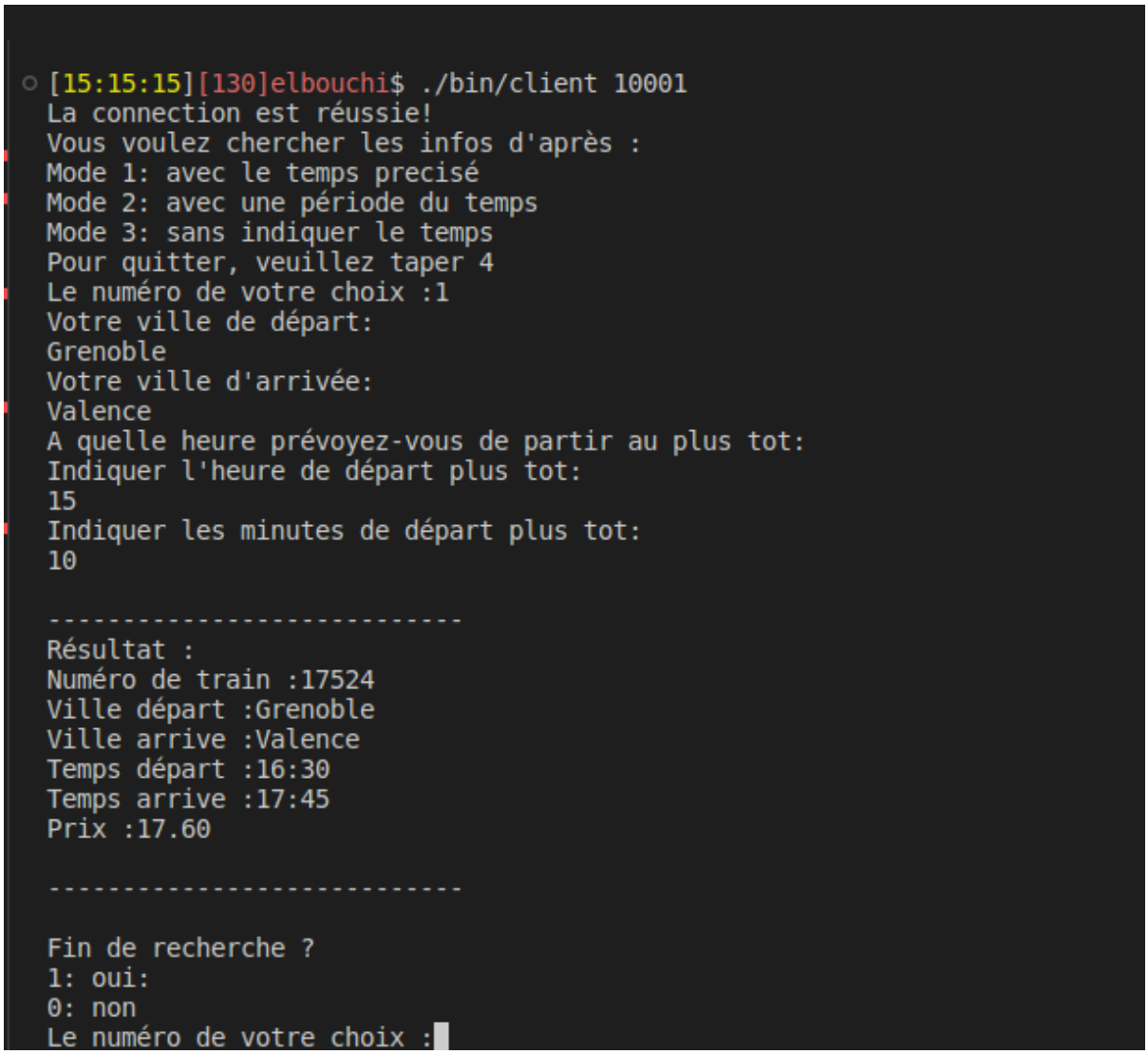
Apres la compilation et l'exécution on obtient comme résultat avec 4 choix si on veut quitter il faut taper 4 et ca va afficher dans le serveur comme quoi le Client est mort:



Ou

Test Fonctionnel :

-Teste de Mode 1 : avec un temps précisé, en spécifiant une plage horaire définie, avec indication de l'heure de départ, qui identifie de manière précise un unique train, le plus proche en réponse à la requête spécifiée (ville de départ, ville d'arrivée, heure de départ).



-Teste de Mode 2: avec une période de temps, en précisant une période de temps déterminée, avec indication de l'heure de départ et une tranche horaire pour l'horaire de départ, qui identifie plusieurs trains dans la plage horaire spécifiée.

```
[15:17:25][130]elbouchi$ ./bin/client 10001
La connection est réussie!
Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :2
Votre ville de départ:
Grenoble
Votre ville d'arrivée:
Valence
A quelle heure prévoyez-vous de partir au plus tot:
Indiquer l'heure de départ plus tot:
14
Indiquer les minutes de départ plus tot:
30
A quelle heure prévoyez-vous de partir au plus tard
Indiquer l'heure de départ plus tard:
18
Indiquer les minutes de départ plus tard:
0
Votre résultat :
1: Le trajet au meilleur prix
2: Le trajet de dureé optimum
Le numéro de votre choix :1

---------------------------
Résultat :
Numéro de train :17524
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :16:30
Temps arrive :17:45
Prix :17.60

---------------------------
Numéro de train :17526
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :17:30
Temps arrive :18:46
Prix :17.60

---------------------------
Numéro de train :17525
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :16:55
Temps arrive :17:55
Prix :17.60
Etat :Supplément
Prix actuellement :19.36
```

Si on veut le trajet au meilleur prix, on tape 1, et on obtient une liste de train avec prix de manier croissant.

```
Pour quitter, veuillez taper 4
Le numéro de votre choix :2
Votre ville de départ:
Grenoble
Votre ville d'arrivée:
Valence
A quelle heure prévoyez-vous de partir au plus tot:
Indiquer l'heure de départ plus tot:
16
Indiquer les minutes de départ plus tot:
40
A quelle heure prévoyez-vous de partir au plus tard
Indiquer l'heure de départ plus tard:
19
Indiquer les minutes de départ plus tard:
1
Votre résultat :
1: Le trajet au meilleur prix
2: Le trajet de dureé optimum
Le numéro de votre choix :2

----------------------------
Résultat :
Numéro de train :17525
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :16:55
Temps arrive :17:55
Prix :17.60
Etat :Supplément
Prix actuellement :19.36

----------------------------
Numéro de train :17528
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :18:30
Temps arrive :19:45
Prix :17.60
Etat :Réduction
Prix actuellement :14.08

----------------------------
Numéro de train :17526
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :17:30
Temps arrive :18:46
Prix :17.60

----------------------------

Fin de recherche ?
```

Si on veut le trajet de durée optimum, on tape 2, et on obtient une liste de train de manière croissante en ce qui concerne la durée de trajet par exemple : le premier résultat la durée du trajet est 1h, le 2eme résultat la durée du trajet est 1h15min le 3eme résultat la durée est de 1h16min.

-Teste de Mode 3: sans indiquer le temps; sans spécifier l'heure, affichant directement tous les trains entre deux villes sans demander d'entrer l'heure de départ et d'arrivée.

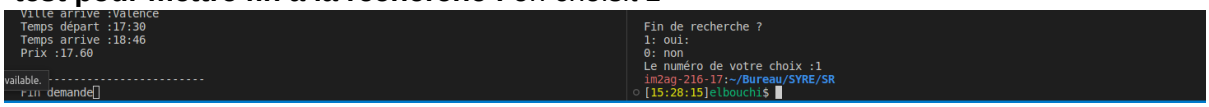
```
Le numéro de votre choix :0

Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :3
Votre ville de départ:
Grenoble
Votre ville d'arrivée:
Valence
Votre résultat :
1: Le trajet au meilleur prix
2: Le trajet de dureé optimum
Le numéro de votre choix :1

----------------------------
Résultat :
Numéro de train :17528
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :18:30
Temps arrive :19:45
Prix :17.60
Etat :Réduction
Prix actuellement :14.08

----------------------------
Numéro de train :17524
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :16:30
Temps arrive :17:45
Prix :17.60

----------------------------
Numéro de train :17526
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :17:30
Temps arrive :18:46
Prix :17.60

----------------------------
Numéro de train :17525
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :16:55
Temps arrive :17:55
Prix :17.60
Etat :Supplément
Prix actuellement :19.36
```

Si on veut le trajet au meilleur prix, on tape 1, et on obtient une liste de train avec prix de manier croissant.

```
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :3
Votre ville de départ:
Grenoble
Votre ville d'arrivée:
Valence
Votre résultat :
1: Le trajet au meilleur prix
2: Le trajet de dureé optimum
Le numéro de votre choix :2

---------------------------
Résultat :
Numéro de train :17525
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :16:55
Temps arrive :17:55
Prix :17.60
Etat :Supplément
Prix actuellement :19.36

---------------------------
Numéro de train :17524
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :16:30
Temps arrive :17:45
Prix :17.60

---------------------------
Numéro de train :17528
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :18:30
Temps arrive :19:45
Prix :17.60
Etat :Réduction
Prix actuellement :14.08

---------------------------
Numéro de train :17526
Ville départ :Grenoble
Ville arrive :Valence
Temps départ :17:30
Temps arrive :18:46
Prix :17.60

---------------------------

Fin de recherche ?
```

Si on veut le trajet de durée optimum, on tape 2, et on obtient une liste de train de manière croissante en ce qui concerne la durée de trajet

-test pour mettre fin à la recherche : on choisit 1



-test pour refaire une recherche : on choisit 0

```
Fin de recherche ?
1: oui:
0: non
Le numéro de votre choix :0

Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :
```

-Si il reste plus de trains dans l'heure de départ indiqué, on aura un message qui indique : pas de train disponible.

```
[14:28:59][141]elbouchi$ ./client 50001
La connection est réussie!
Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Le numéro de votre choix :2
Votre ville de départ:
Grenoble
Votre ville d'arrivée:
Valence
Votre temps de départ:
Indiquer l'heure de départ:
20
Indiquer les minutes de départ:
2
Votre temps d'arrivée:
Indiquer l'heure d'arrivée:
21
Indiquer les minutes d'arrivée:
0
Votre résultat :
1: Le trajet au meilleur prix
2: Le trajet de dureé optimum
Le numéro de votre choix :1

---------------------------
Résultat :
Pas de train disponible.
---------------------------

Fin de recherche ?
1: oui:
0: non
Le numéro de votre choix :
```

Test robustesse :

-Teste de Mode 1:

```
Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :55
Le numéro de votre choix :88
Le numéro de votre choix :ff
Le numéro de votre choix :1
Votre ville de départ:
jjtytj
Votre ville d'arrivée:
uoouout
A quelle heure prévoyez-vous de partir au plus tot:
Indiquer l'heure de départ plus tot:
11
Indiquer les minutes de départ plus tot:
11

----------------------------
Résultat :
Pas de train disponible.
----------------------------

Fin de recherche ?
1: oui:
0: non
Le numéro de votre choix :
```

Si on indique des ville qui n'existe pas dans le fichier .txt ; on aura un message d'erreur qui nous informe comme quoi il n'existe pas de train disponible

-Teste de Mode 2:

```
Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :2
Votre ville de départ:
sdgsh
Votre ville d'arrivée:
iopiiop
A quelle heure prévoyez-vous de partir au plus tot:
Indiquer l'heure de départ plus tot:
ureu
Indiquer l'heure de départ plus tot:
ueru
Indiquer l'heure de départ plus tot:
uteueu
Indiquer l'heure de départ plus tot:
11
Indiquer les minutes de départ plus tot:
22
A quelle heure prévoyez-vous de partir au plus tard
Indiquer l'heure de départ plus tard:
55
Indiquer l'heure de départ plus tard:
99
Indiquer l'heure de départ plus tard:
99
Indiquer l'heure de départ plus tard:
88
Indiquer l'heure de départ plus tard:
55
Indiquer l'heure de départ plus tard:
60
Indiquer l'heure de départ plus tard:
2
Indiquer les minutes de départ plus tard:
15
Votre résultat :
1: Le trajet au meilleur prix
2: Le trajet de dureé optimum
Le numéro de votre choix :1

----------------------------
Résultat :
Pas de train disponible.
----------------------------

Fin de recherche ?
1: oui:
0: non
Le numéro de votre choix :
```

Si l'on ne renseigne pas correctement les heures et les minutes, le programme affichera continuellement le même message jusqu'à ce qu'une heure et une minute valides soient fournies. Par exemple, si l'utilisateur entre une chaîne de caractères ou une heure invalide (comme 55), le programme continuera de redemander jusqu'à ce qu'une heure et une minute valides soient correctement saisies.

-Teste de Mode 3:

```
Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :3
Votre ville de départ:
ytrkrt
Votre ville d'arrivée:
uitup
Votre résultat :
1: Le trajet au meilleur prix
2: Le trajet de dureé optimum
Le numéro de votre choix :2

---------------------------
Résultat :
Pas de train disponible.
---------------------------

Fin de recherche ?
1: oui:
0: non
Le numéro de votre choix :
```

-Test des modes:
Si on choisit pas les valeurs de 1 à 4 ca redemande toujours de rentrer le numero de choix.

```
Vous voulez chercher les infos d'après :
Mode 1: avec le temps precisé
Mode 2: avec une période du temps
Mode 3: sans indiquer le temps
Pour quitter, veuillez taper 4
Le numéro de votre choix :55
Le numéro de votre choix :88
Le numéro de votre choix :ff
Le numéro de votre choix :
```

En réponse aux questions posées par le professeur vendredi, nous avons apporté les modifications suivantes :

- Comme la fonction `read` peut écouter le pipeline de communication entre deux serveurs, nous avons établi une condition `if` pour la valeur de retour de la fonction `read`. Si l'utilisateur saisit `control c` à n'importe quel endroit où une entrée est requise, le pipe entre les serveurs sera fermé et la fonction `read` retournera `0`. Dans

ce cas, le message `serveur mort` sera affiché et le programme se terminera correctement.

- Concernant le cas où `mode1` ne retourne qu'un seul horaire de cours précis tout en demandant à l'utilisateur de choisir un tri, nous avons ajusté le contenu affiché sur le client pour le rendre plus logique.