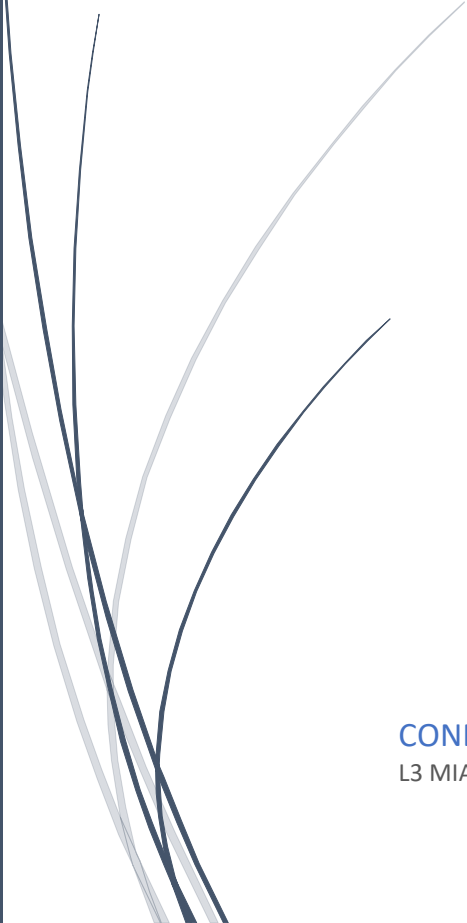


A thick dark blue vertical bar is on the left. A blue arrow points right from it, containing the date.

02/12/2022

RAPPORT DU PROJET SYSTEME/RESEAUX

Application client/serveur TCP/IP

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

CONDE Seynabou – JANDOT Floriane – KONE Zeïnabou Bissi
L3 MIAGE – UGA

I. INTRODUCTION/ MISE EN CONTEXTE

Ce rapport est un récapitulatif des réalisations effectuées au cours du projet de Système/Réseaux. Il a été réalisé en **5 phases** de façon incrémentale.

Ce projet consistait à la mise en place d'une **application client/serveur** inspiré par le protocole **TCP/IP**. L'idée repose sur la réalisation d'un serveur que l'on peut interroger pour obtenir des informations. Pour faire court, les clients demandent des informations (via des requêtes) au serveur, qui va se charger de les rechercher dans un fonds documentaire. Une fois qu'il détient les informations souhaitées par les clients (si elles existent), le serveur les lui renvoie. Il s'agit de requêtes du type : ***Peux-tu me donner les informations du livre dont l'auteur est ***, qui a comme référence***....***

Nous allons détailler, ci-après, les phases d'implémentations des requêtes, les choix concernant les protocoles d'échanges et tout ce dont nous avons eu besoin pour faire le projet.

II. PHASE 1

Cette première phase du projet consistait à implémenter le **noyau de client/serveur TCP/IP**.

Nous avons suivi les indications du cours et du petit mémento qui nous a été fourni pour l'implémenter.

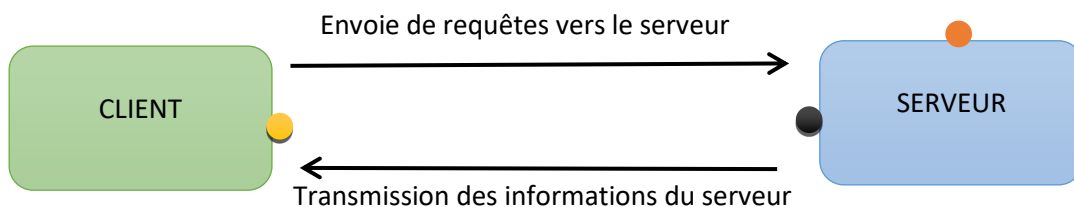


Figure A

La **Figure A** ci-dessous illustre la façon (très élémentaire) dont la communication entre le serveur et le(s) client(s) va se passer. Cette communication se déroule comme suit :

- Le serveur va se mettre en écoute sur un socket d'écoute qu'il aura créée (petit rond orange ● de la **Figure A**) dans l'attente d'une éventuelle connexion de la part d'un client ;
- Une fois qu'une connexion est établie, le client est donc en mesure d'envoyer des requêtes au serveur via le socket du client (petit rond jaune ● de la **Figure A**) ;
- Lorsque le serveur reçoit la requête, il effectue le traitement qu'il faut et renvoie les informations souhaitées au client via son socket de service (petit rond noir ● de la **Figure A**) ;
- L'échange serveur/client continu jusqu'à ce qu'il y ait une rupture de connexion du client.

III. PHASE 2

Cette deuxième phase du projet est la **plus importante** : ici, nous avons poussé notre réflexion sur comment assurer la communication entre le serveur et les clients mais également quelle(s) information(s) circule(nt) dans le noyau et à quel moment. Le schéma ci-dessous illustre notre réflexion et la façon dont la communication a été gérée.

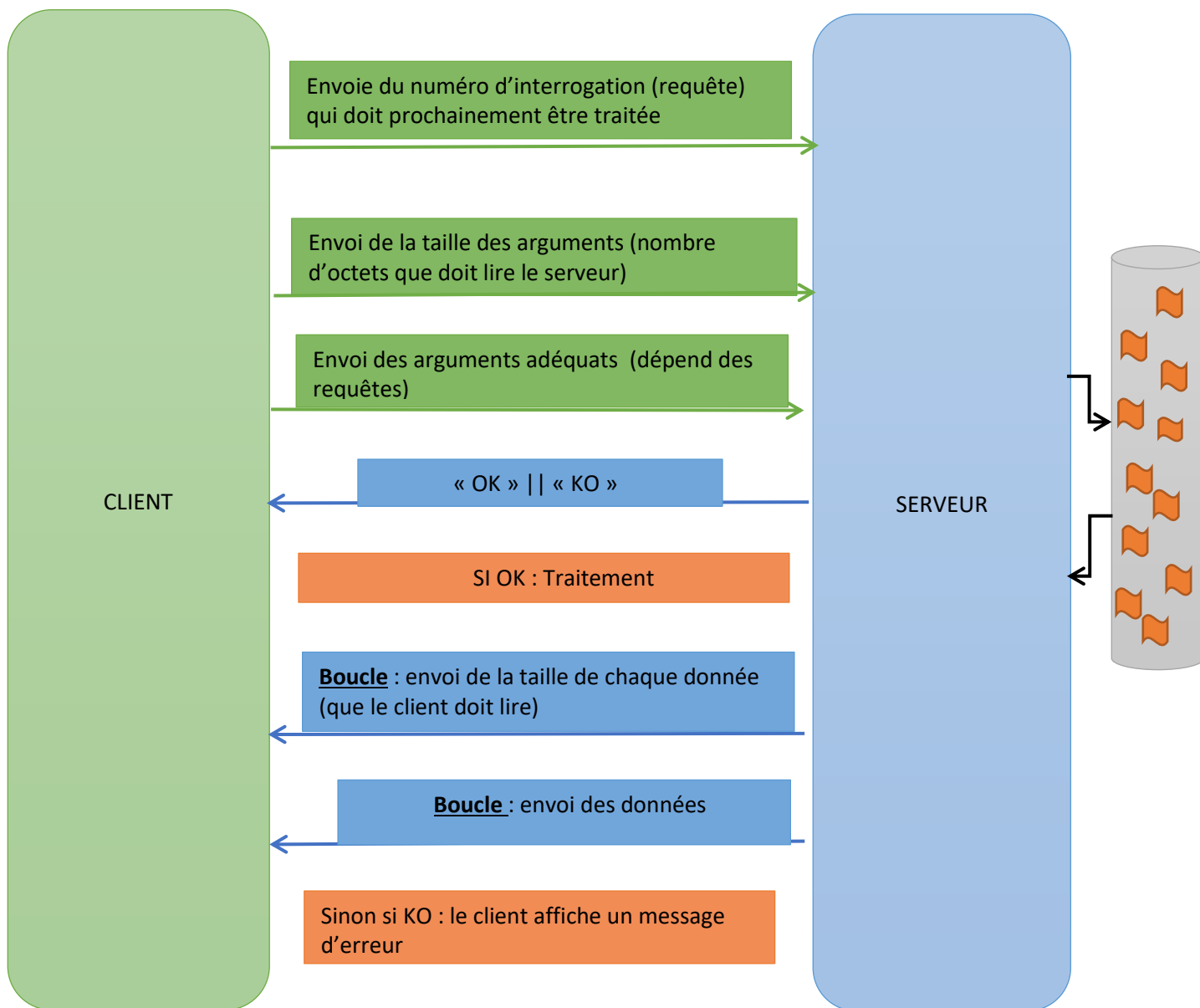


Figure B

❖ Organisation et protocole d'échange

Lorsque la connexion est établie entre le client et le serveur, l'utilisateur a la possibilité de faire un choix en fonction de la requête qu'il veut faire émettre au serveur. Ce choix est représenté par l'entier $i \in \llbracket 1,4 \rrbracket$ avec i le numéro de la requête. Le client va donc envoyer ce numéro de requête au serveur. L'utilisateur émet ensuite un ou plusieurs arguments. En fonction de cette dernière, le client

se charge également de transmettre au serveur la taille des arguments entrés au clavier puis les arguments eux-mêmes. Si la requête peut être traitée, le serveur envoie « **OK** » au client qui va lire cette information et attendre la réception des informations souhaitées. Le serveur traite alors la requête (cherche les informations souhaitées dans le fonds documentaire) puis envoie, séparément, la taille de toutes les données qu'il a pu traiter et enfin les données en elles-mêmes (séparément). Une fois toutes les données reçues par le client (les unes après les autres), il se charge de les formater et de les afficher à l'écran pour l'utilisateur.

Dans les explications à venir, nous détaillerons de façon concise et profonde le déroulement de chaque type de requête et la façon dont la communication a été adaptée pour chacune d'elle. Nous détaillerons l'implémentation des primitives réalisées un peu plus loin dans le rapport (PHASE 4).

IV. PHASE 3

a. Mise en contexte

Cette phase a consisté à la réalisation des primitives pour chaque requête. Comme énoncé un peu plus haut, nous avons mis en place un menu qui permet à l'utilisateur de choisir le type de requête (**interrogation**) qu'il veut émettre au serveur :

- La saisie de l'entier « **1** » correspond à la **première interrogation** : cette interrogation/requête consiste à soumettre une référence (un entier) au serveur qui se chargera de rechercher les informations associées au livre qui est référencé par cet entier. Si la référence n'existe pas, le client se charge d'informer l'utilisateur en affichant un message précis qui l'explique.
- La saisie de l'entier « **2** » correspond à la **deuxième interrogation** : Il s'agit de soumettre des mots-clés (plusieurs arguments). Le serveur se chargera de rechercher la présence de ces mots-clés dans le titre de chaque livre du fonds documentaire. Pour tous les livres qui contiennent au moins un de ces mots-clés, le serveur récupère un certain nombre d'informations les concernant (référence, nom de l'auteur, titre et genre) et les envoie au client. Le client se charge de soumettre les informations trouvées à l'utilisateur en les affichant par ordre alphabétique du nom de l'auteur. Et si toutefois aucun des mots-clés soumis ne se trouve dans aucun titre des livres, le client délivre un message clair qui va dans ce sens.
- La saisie de l'entier « **3** » correspond à la **troisième interrogation** : ce cas est similaire à l'interrogation n°2, à la seule différence du fait que nous avons exactement deux arguments et que la recherche s'effectue cette fois-ci dans les noms d'auteur et dans les genres de tous les livres.
- La saisie de l'entier « **4** » correspond à la **dernière interrogation** : ici, il s'agit de soumettre au serveur un nom d'auteur. Ce dernier va se charger de rechercher tous les livres de cet auteur et ne renvoyer que celui qui est bien noté (A).
- Pour pouvoir manipuler le fonds documentaire, nous avons fractionner notre *fichier.txt* pour le stocker les livres dans un tableau comme illustré ci-dessous :

Livre1	Livre2	Livre3	Livre4	Livre5	Livre6	Livre7	Livre8	Livre9	Livre10
--------	--------	--------	--------	--------	--------	--------	--------	--------	---------

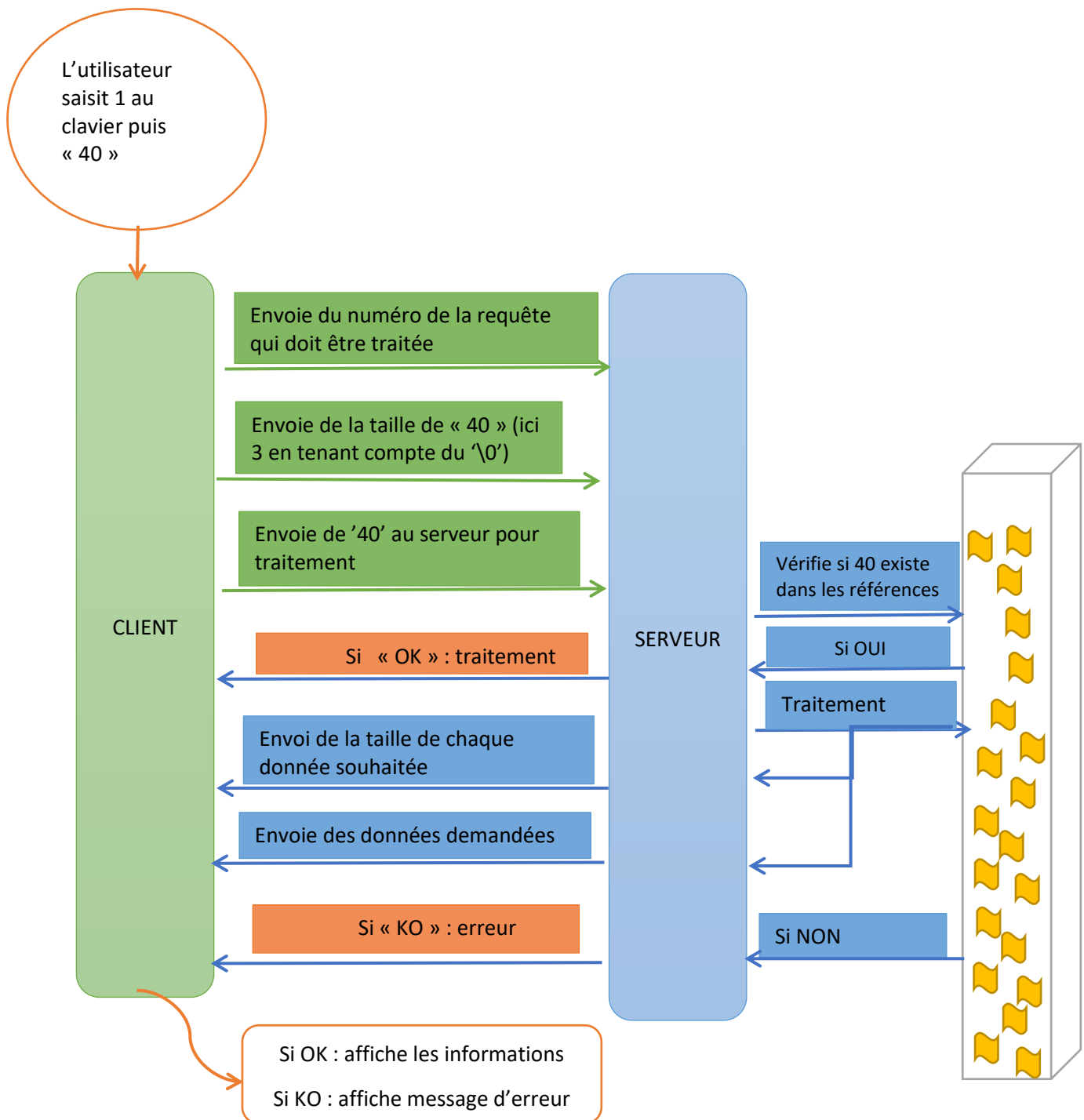
Ici, chaque livre est également un tableau de chaînes de caractères qui contient les informations comme la référence, le nom d'auteur, le titre du livre, le genre littéraire, le nombre de pages et l'appréciation :

Référence	Nom d'auteur	Titre du livre	Genre	Nb Pages	Appréciation
-----------	--------------	----------------	-------	----------	--------------

Chaque case du tableau précédent est également un tableau (car chaîne de caractères). Finalement, nous avons un tableau à trois dimensions que nous nommerons **tab** pour les explications dans la suite.

b. Requête 1 : Soumission d'une référence

Le schéma ci-dessous illustre cette interrogation ainsi que son protocole d'échange.



Synthèse du schéma :

Cercle et rectangle au contour orange : UTILISATEUR

Bloc de couleur verte : CLIENT

Bloc de couleur bleue : SERVEUR

Carton vertical grisâtre : FONDS DOCUMENTAIRE

Flèches vertes : envois du client vers le serveur

Flèches bleues : envois du serveur vers le client

Flèches oranges : Interaction Client/Utilisateur

Comme on peut le voir sur le schéma précédent, une fois la connexion établie entre le client et le serveur, le menu de choix est présenté à l'utilisateur. Dans ce cas précis, si l'utilisateur saisit l'entier « 1 », il lui sera donc demandé de saisir la référence dont il souhaite avoir les informations du livre concerné. Comme exemple, nous avons pris 40. Une fois qu'il saisit la référence, le client se charge d'envoyer au serveur la taille de cet argument. Quand le serveur aura lu cette information, le client lui transmet donc '40'. Lorsque le serveur aura lu cette nouvelle information (numéro de référence), il recherche sa présence dans les références de tous les livres du fonds documentaire (tous les `tab[i][0]`).

- S'il trouve le livre concerné, il envoie « OK » au client (qui va lire cette information et attendre les données). Le serveur envoie ensuite au client la taille des données demandées. Le client va à son tour lire cette deuxième information. Puis, le serveur lui envoie le **nom de l'auteur**, puis le **titre du livre**, et ensuite le **genre** et le **nombre de pages**. Attention, sachant que les informations sont envoyées séparément par le serveur, le client les lit également les unes à la suite des autres. Une fois que le client a fini de lire toutes les données envoyées par le serveur, il se charge de les formater et les afficher pour l'utilisateur (de façon claire).
- S'il ne trouve pas un livre référencé par le numéro envoyé, il informe le client en lui transmettant « KO » et ce dernier va se charger d'informer l'utilisateur en affichant un message du type « **la référence demandée n'existe pas** ».

Les autres formes de requêtes suivent la même logique (schéma). Il n'y a que les arguments qui changent.

c. Requête 2 : Soumission de mots-clés

Dans ce deuxième cas de requête, l'utilisateur va soumettre des mots-clés. Tout d'abord, il va saisir « 2 » au clavier (*pour informer le client du type de requête qu'il souhaiterait*). Le client va indiquer ce choix au serveur. Puis, il saisit les mots-clés. Le client va alors se charger d'envoyer un bloc de chaînes de caractères (tableau contenant les arguments) au serveur (*en prenant garde d'envoyer **en premier lieu** la taille de ce bloc d'arguments*). Le serveur à son tour va fractionner ce tableau en sous-chaînes (*chaque sous-chaîne étant un mot-clé*). Il va par la suite rechercher la présence **d'au moins un de ces mots-clés dans les titres** de tous les livres du fonds documentaire.

- S'il trouve, dans un ou plusieurs titres au moins un mot-clé, il va envoyer « OK » au client (qui lira cette information dans l'attente des données). Puis, il va trier les informations du/des livre(s) concerné(s) en fonction de la demande soumise par le client (**référence, nom-auteur,**

titre du livre et genre). Une fois qu'il a réussi à récupérer ces données, il envoie leurs tailles au client pour ensuite les envoyer (*séparément et de façon successive dans cet ordre : référence – nom de l'auteur – titre – genre*) au client (qui va les lire les unes à la suite des autres). Sachant que l'on peut avoir plusieurs livres qui satisfont à cette requête, le serveur va se charger de les transmettre au client en prenant soin de faire cette transmission par ordre alphabétique des noms d'auteurs. Le client, à la fin va se charger d'afficher les informations qu'il a lues à l'écran pour l'utilisateur.

- Si aucun livre ne satisfait la requête (c'est-à-dire qu'aucun des mots-clés n'est présent dans aucun des titres des livres), il renvoie « KO » au client qui en déduira la non-existence. Il affiche donc à l'écran un message du type « **la requête n'a pas pu être exécutée** ».

CLIENT :

- Envoie le numéro de la question à traiter au serveur
- Récupère les arguments saisis par l'utilisateur et les stocke dans un tableau
- Envoie la taille des arguments au serveur
- Envoie le tableau d'arguments au serveur
- Si le serveur lui envoie OK :

Il saura qu'il doit s'attendre à recevoir des informations

Il va lire d'abord la taille des informations qui lui sont envoyées et ensuite les récupérer et les afficher à l'écran

- S'il reçoit KO de la part du serveur : il en déduit que la requête ne peut pas être traitée et affiche un message d'erreur

SERVEUR :

- Lit la taille des arguments envoyés (qu'il doit lire)
- Lit et fractionne le tableau d'arguments en sous-chaînes (mots-clés)
- Recherche la présence d'au moins un mot-clé dans les titres
- Si PRESENCE :
 - Envoie OK au client
 - récupère les livres qui satisfont à la requête et trie les informations voulues
 - Envoi de la taille des informations au client
 - Envoi de ces informations par ordre alphabétique des noms d'auteurs au client
- Si NON PRESENCE : renvoie KO au client

d. Requête 3 : **Soumission du nom d'un auteur et du genre du livre**

Ce troisième cas est similaire au cas précédent, car elle consiste aussi à la soumission de mots-clés. Mais ici, il y'a plus de contraintes : il faut impérativement que le **premier mot** saisi soit le **nom**, et donc le **deuxième le genre**. Et nous n'avons exactement que deux mots-clés (pas moins et pas plus). De façon analogue à la requête 2, le client va envoyer la taille des arguments puis le bloc d'arguments en lui-même (cette fois-ci de taille fixe = 2) au serveur, qui va fractionner ce bloc en deux sous-chainés. Il va ensuite rechercher la présence de la première sous-chaine (**nom**) dans tous les noms d'auteurs (**tab[i][1]**) et celle de la deuxième sous-chaine (**genre**) dans tous les genres (**tab[i][3]**).

Note : ici, il est possible de mettre le **nom complet** de l'auteur ou juste une **partie**. Par exemple, que l'utilisateur saisisse **Victor ou Victor Hugo**, le serveur est capable de traiter la requête.

- ❖ S'il trouve un/des livre(s) qui satisfait/satisfont à cette requête, il va envoyer « OK » au client (comme de coutume, ce dernier va lire cette information dans l'attente des autres). Puis, il va récupérer les informations associées au(x) livre(s) disponible(s) pour cet auteur et ce genre. Une fois qu'il a réussi à récupérer ces données, il envoie leurs tailles puis les elles-mêmes (*séparément et de façon successive dans cet ordre : référence – nom de l'auteur – titre – genre*) au client (qui va les lire les unes à la suite des autres). Le client quant à lui va lire ces données, les formater et les afficher à l'écran pour l'utilisateur.
- ❖ Si aucun livre ne satisfait la requête (c'est-à-dire qu'aucun des mots-clés n'est présent dans aucun des titres des livres, il renvoie « KO » au client qui en déduira la non-existence. Il affiche donc à l'écran un message du type « **la requête n'a pas pu être exécutée** ».

CLIENT :

- Envoie le numéro de la requête à traiter au serveur
- Récupère le nom d'auteur et le genre dans un tableau d'arguments (nom d'abord et ensuite genre)
- Envoie la taille de ces arguments au serveur
- Envoie le tableau d'arguments au serveur
- Si le serveur lui envoie OK :

Il saura qu'il doit s'attendre à recevoir des informations

Il va lire d'abord la taille des informations qui lui sont envoyées et ensuite les récupère et les affiche à l'écran

- S'il reçoit KO de la part du serveur : il en déduit que la requête ne peut pas être traitée affiche un message d'erreur

SERVEUR

- Récupère la taille des arguments (qu'il doit lire)
- Lit et fractionne le tableau d'arguments en sous-chainés (nom et genre)
- Recherche la présence du nom dans les noms d'auteurs de tous les livres ET celle du genre pour cet auteur
- Si PRESENCE :
- Envoie OK au client
- récupère les informations associées
- Envoie la taille des informations au client
- Envoie des informations au client
- Si NON PRESENCE : Envoie de « KO » au client

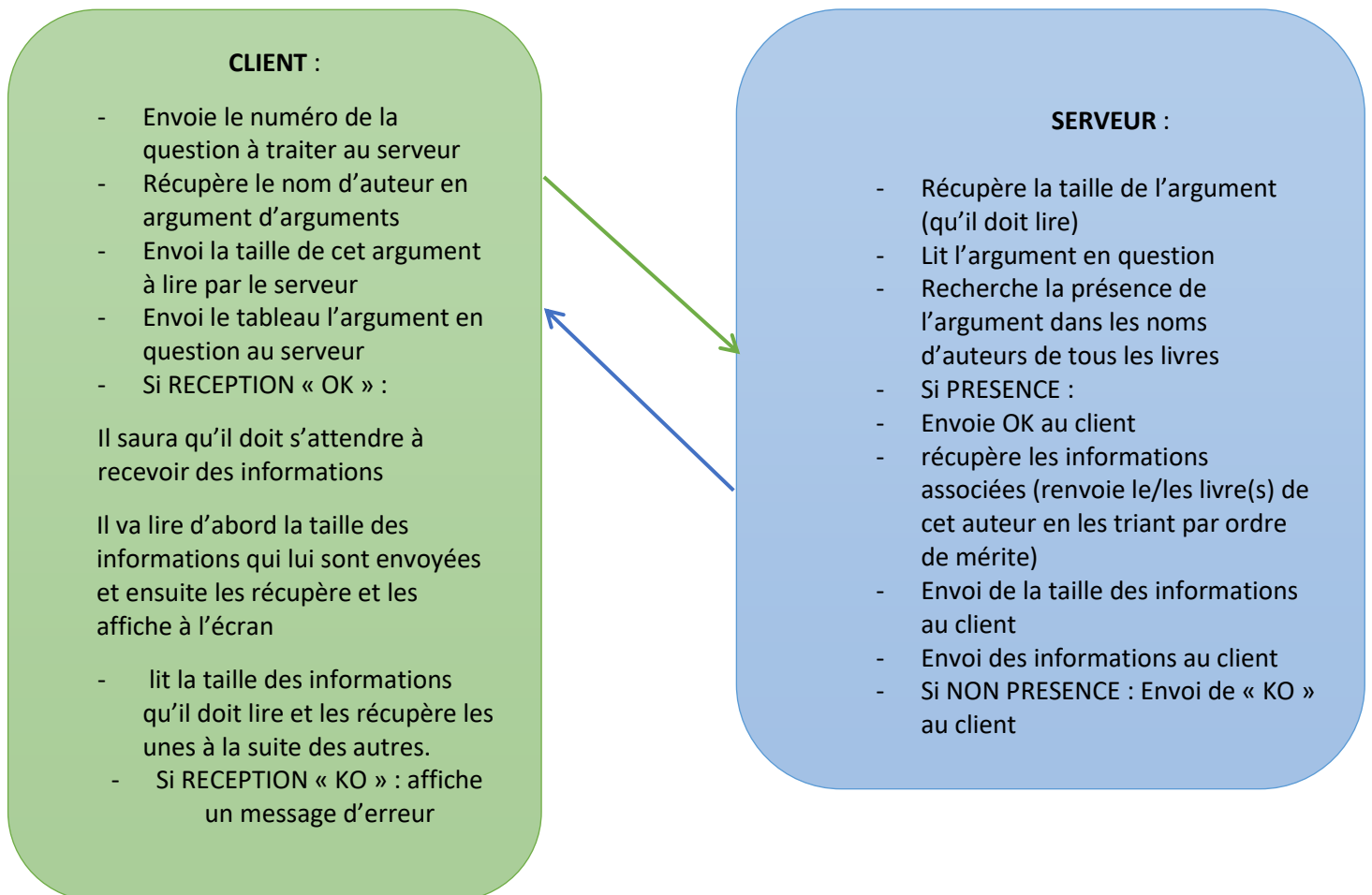
e. Requête 4 : Soumission du nom de l'auteur pour récupérer son/ses livre(s) le/les mieux noté(s)

A ce stade, nous avons la dernière requête qui consiste à soumettre au serveur **le nom d'un auteur**.

L'utilisateur commence par saisir l'entier « 4 » au clavier, puis le nom de l'auteur souhaité. Le client va donc se charger d'envoyer d'abord la taille de cet argument au serveur puis l'argument en lui-même (le serveur va donc lire d'abord la taille envoyée par le client puis l'argument). Une fois la réception de l'argument (lecture) par le serveur, il va rechercher la présence de ce nom d'auteur dans tous les noms d'auteurs de tous les livres (**tab[i][1]**).

Note : Contrairement au cas de la requête 3, ici il **faut** saisir le **nom complet** de l'auteur ; sinon le serveur n'est pas capable de traiter la requête.

- S'il trouve une correspondance, il va envoyer « **OK** » au client. Ensuite, il va rechercher tous les livres de cet auteur dans le fonds documentaire. Une fois en possession de ces données, il va classer les livres en par ordre de mérite. La taille de ses informations sera donc envoyée au client et s'ensuivra les informations elles-mêmes. Une fois que le client a pu lire (de façon successive et séparément) les deux dernières informations qui lui ont été envoyées, il se chargera de les formater et de les afficher à l'écran.
- S'il ne trouve pas de correspondance, le serveur enverra « **KO** » au client. Lorsque ce dernier aura lu l'information, il affichera un message d'erreur du type « **ce nom d'auteur n'existe pas** ».



V. PHASE 4

Ces deux dernières phases du projet ont consisté à intégrer toutes nos fonctionnalités de bases avec le noyau client/serveur.

Ci-dessous, nous allons donc détailler la conception de toutes les fonctions que nous avons réalisées

Pour le serveur :

Lorsque le serveur reçoit un ou des arguments de la part du client (c'est-à-dire lorsqu'il lit ce qui lui a été envoyé), il recherche directement la présence de ce/ces argument(s) dans le fonds documentaires (qui dépend du type de requête). S'il trouve une correspondance, il doit envoyer « OK » au client, sinon il enverra « KO ». Cette idée est réalisée dans la fonction **verification_ok_ko** qui prend en paramètre le socket de service du serveur (socket par lequel il recevra les arguments de la part du client) et le/les argument(s) en question. En somme, les « *signaux* » OK et KO sont juste des indications qui permettent au client de savoir s'il recevra des informations après ou pas.

Pour que le serveur puisse **recevoir** les arguments venant du client, nous avons implémenté la fonction **reception_donnees** qui assure la lecture des informations : le serveur commence par lire le numéro d'interrogation qu'il sera amené à traiter, puis il lit la taille des arguments qui vont lui être donnés et enfin les arguments eux-mêmes.

Pour qu'il puisse envoyer les informations qu'il aura traitées au client, nous avons la fonction **envoi_donnees** qui nous permettra d'écrire dans le socket de service : le titre, l'auteur, le genre et le nombres de pages des livres.

Nous avons également des fonctions intermédiaires qui nous permettent de gérer faire nos tris, fractionner nos arguments ...

Par la suite, nous avons réalisé des fonctions (de recherche des informations) adaptées pour chaque requête :

- **Pour la requête 1** (soumission d'une référence), on a la fonction **recherche_ref** qui va rechercher la référence soumise dans la première colonne (colonne des numéros de référence) du tableau de livres et va renvoyer le livre (une ligne du *fichier.txt*) dont la référence correspond à celle qui a été soumise.
- **Pour la requête 2** (soumission de mots-clés), nous avons la fonction **recherche_mots_cles** qui va rechercher la présence d'au moins un des arguments reçus par le serveur dans la troisième colonne (colonne des titres des livres) et va renvoyer tous les livres (toutes les lignes du fichier *fichier.txt*) dont le titre contient au moins un de ces mots-clés. Sachant que nous pouvons avoir plusieurs livres qui satisfont à cette requête, le serveur va les envoyer par ordre alphabétique des noms d'auteurs au client.
- **Pour la requête 3** (soumission de nom et de genre), nous avons la fonction **recherche_nom_genre** qui est assez similaire à la précédente. Au lieu d'effectuer la recherche dans les titres, elle va plutôt l'effectuer dans tous les noms d'auteurs ET les genres littéraires (deuxième et quatrième colonne du fichier *fichier.txt*) et va renvoyer tous les livres de ce genre pour cet auteur.
- **Pour la dernière requête qui est la 4** (soumission de nom d'auteur), on a la fonction **recherche_nom_critere** qui va rechercher tous les livres (toutes les lignes du *fichier.txt*) dont l'auteur est le nom qui a été soumis au serveur et les renvoyer en les triant par ordre de mérite.

Nous avons réalisé, pour chaque type de requête une fonction qui assure son traitement :

- ❖ **fonctionnalite 1** : cette fonction assure le traitement de la requête 1 (soumission d'un numéro de référence). Lorsqu'elle est appelée, elle réserve de la mémoire pour un tableau de string : ce tableau contiendra un livre (c'est à dire un titre, un auteur...). La fonction appelle ensuite **recherche_ref** qui remplit le tableau créé du livre trouvé. Enfin, elle vérifie si le contenu du tableau : s'il est vide ("ko"), cela veut dire qu'aucun livre ne correspond à la requête (il envoie donc KO au client). Au contraire, s'il contient au moins un élément ("ok"), il va envoyer OK au client et on appelle la fonction **envoi_donnees** qui écrit les données pour que le client les lise.
- ❖ **fonctionnalite 2** : cette fonction assure le traitement de la requête 2 (soumission de mots-clés). Dans un premier temps, elle réserve la mémoire d'un tableau de string puis y place les arguments envoyés **séparés par un espace** par la fonction **fractionner_espace**. Exemple : le client envoie "chartreuse contemplation" --> le tableau contient {"chartreuse", "contemplation"}. Elle réserve ensuite la mémoire d'un double tableau de string qui va contenir cette fois une **liste** de livres, puis le remplit par la fonction **recherche_mot_cles**. Si le tableau ne contient pas de résultat ("ko"), aucun livre ne correspond à ces mots clés (le serveur envoie donc KO au client). Sinon si le tableau contient au moins un résultat, le serveur envoie OK au client et la fonction lui écrit le nombre de livres trouvés afin de pouvoir boucler sur l'envoi des données.
- ❖ **fonctionnalite 3** : cette fonction assure le traitement de la requête 3 (soumission de nom d'auteur et de genre). La fonction réserve d'abord un tableau de string pour séparer les arguments, à la manière de la 2. Mais cette fois-ci, nous avons exactement deux arguments, avec la possibilité d'avoir **au moins** deux mots pour le nom de l'auteur. En effet pour un auteur, nous pouvons avoir juste **un nom**, un **prénom-nom**, voire un **nom composé** ! Nous devons donc partir dans la théorie que tous les mots représentent le nom de l'auteur à l'exception du dernier qui est le genre. Pour coder cette idée, on réserve un deuxième tableau de string (à deux cases) et on y place : - dans la première case, la concaténation des x premiers mots (on n'oublie pas d'ajouter un espace entre chacun d'eux), dans la deuxième case, le dernier mot qui est le genre exemple : ["Victor" "Hugo" "roman"] --> le tableau contient ["Victor Hugo" "roman"]. Enfin, on réserve la mémoire d'un double tableau de String où on placera les livres retournés via la fonction **recherche_nom_genre**. La fonction écrit le nombre de livres trouvés puis envoie les données si "ok", et n'écrit rien si "ko" (comme pour les autres cas, les signaux d'indication OK ou KO sont envoyés au client avant que le serveur ne fasse l'envoi des données, s'il y en a).
- ❖ **Fonctionnalite 4** : cette fonction assure le traitement de la requête 4 (soumission de nom d'auteur). La fonction ressemble aux précédentes : elle réserve de la mémoire pour un double tableau et le remplit via **recherche_nom_critere**. Elle écrit le nombre de livres trouvés et envoie les données si "ok", ne fait rien si "ko".

Remarque : Attention à ne surtout pas mettre un espace après la saisie des arguments. Le serveur ayant reçu la taille des arguments qu'il doit lire avant la réception de ces derniers, il ne reconnaîtra pas l'argument s'il y a un caractère de plus (caractère espace).

Toutes ces implémentations se trouvent dans le fichier **traitement_serveur.c**.

Pour le client :

En ce qui concerne le client, nous avons implémenté une fonction **communication_serveur** qui va gérer la communication du client avec le serveur. Nous avons un petit menu pour permettre l'interaction avec l'utilisateur en fonction du type de requête qu'il souhaite faire exécuter. Si le numéro de question est différent de 1, 2, 3 ou 4, le client affiche à l'écran un message d'erreur du type « la question demandée n'existe pas ». Si non, en fonction du numéro de la question choisi, il indique au serveur que c'est « *telle requête* » qu'il sera amené à traiter. Ensuite, lorsque l'utilisateur aura saisi les arguments (qui dépend de la requête), le client va d'abord envoyer la taille des arguments (pour indiquer au serveur combien d'octets il doit lire) puis les arguments eux-mêmes.

Il lira ensuite le signal qui lui sera envoyé par le serveur (OK ou KO). S'il reçoit OK, il va donc s'attendre à recevoir des données. En fonction du numéro de requête qui sera traitée par le serveur, le client va lire les bonnes informations au bon moment (**switch case du code**). Sinon si c'est l'indication KO qu'il reçoit, il affichera un message d'erreur du type « *la référence donnée n'existe pas* » ou encore « *Aucun livre n'est de cet auteur et de ce genre* » par exemple.

Le client a également la charge du formatage et de l'affichage comme nous avons pu le souligner dans les précédentes explications. A ce niveau, nous avons implémenté la fonction **lecture_informations** qui permet au client de lire les informations qu'il reçoit du serveur et de les afficher une à une (sachant que tous les envois et réceptions de données se font séparément).

NB : Nous aurions voulu implémenter une fonction de lecture et d'affichage d'informations propre à chacune des requêtes. Par exemple pour la requête 1, il nous est demandé de ne renvoyer que le nom de l'auteur, le genre et le titre du livre qui la satisfait. Mais comme vous avez pu le voir, nous affichons les mêmes informations pour toutes les requêtes, à savoir : le nom de l'auteur, le titre, le genre, le nombre de pages (> ou < à 300) et l'appréciation.

Titre : les misérables

Nom d'auteur : Victor Hugo

Genre : roman

Page > 300

Note : A

Ceci est une piste d'amélioration que nous aurions voulu explorer. En somme, pour chaque requête, on aurait fait des fonctions qui est adaptée aux informations demandées.

Nous avons aussi une fonction **compter_mot** qui n'est utilisée que pour la *fonctionnalité_3* : elle permet de vérifier que nous avons exactement deux arguments passés en paramètre : à savoir le nom et le genre. Si on a un mauvais nombre d'arguments ou que par malheur nous inversion les arguments (c'est-à-dire que l'utilisateur saisit le genre avant le nom), le client affichera un message d'erreur.

Toutes ces fonctionnalités sont écrites dans le fichier **traitement_client.c**

IMPORTANT : Les arguments doivent être saisis (pour toutes les requêtes) exactement comme ils sont dans le fichier.txt (les minuscules et les majuscules ne sont pas gérées). Il faut donc tâcher à bien mettre les lettres majuscules et minuscules au bon endroit !!

GESTION DES CAS D'ERREURS

Nous avons pu gérer pas mal de cas d'erreur :

- Si un numéro de question, autre que 1, 2, 3 et 4 est saisi par l'utilisateur, on affiche un message d'erreur (même
- Si les mauvais arguments sont saisis (et pour les requêtes, et pour le lancement de l'exécution, nous avons également un message d'erreur ;
- Lorsque le serveur ne reconnaît pas le fichier qui lui est passé en argument, nous avons également un message d'erreur qui s'affiche.

VI. PHASE 5

Dans cette toute dernière phase, nous avons réorganisé nos fichiers. Nous avons écrit les fichiers d'en-tête **.h** (**traitement_client.h** et **traitement_serveur.h**) qui se trouvent dans le dossier **headers**.

Les programmes principaux du client et du serveur (qui vont permettre l'exécution du projet) sont dans les fichiers **client.c** et **serveur.c**. Tous nos fichiers sources se trouvent dans le répertoire **sources**.

Nous avons un répertoire **bin** dans lequel se trouvera nos deux exécutables serveur et client.

Ces exécutables sont créés par la compilation avec un **Makefile** qui se trouve également dans le dossier du projet.

COMPILATION ET EXECUTION

Pour compiler le projet, il faudra taper la commande **make** depuis la racine du dit-projet. Si nous souhaitons effacer les fichiers objets qui ont été créés, il faut taper la commande **make clean**.

Pour lancer le serveur, il faut taper la commande **bin/./serveur NOPORT doc/fichier.txt** et pour le client ça sera **bin/./client NOPORT nom_machine_serveur**.

VII. CONCLUSION

En conclusion, nous avons pu implémenter toutes les fonctionnalités de bases de toutes les requêtes. Comme points d'amélioration, nous aurions voulu gérer la sensibilité à la casse des arguments, pouvoir implémenter des fonctions de lecture d'informations et d'affichages propres à chaque requête mais aussi pouvoir gérer le cas où une chaîne vide est entrée en argument.