Oukkal Yacine

Consultation et stockage d’un dictionnaire de données

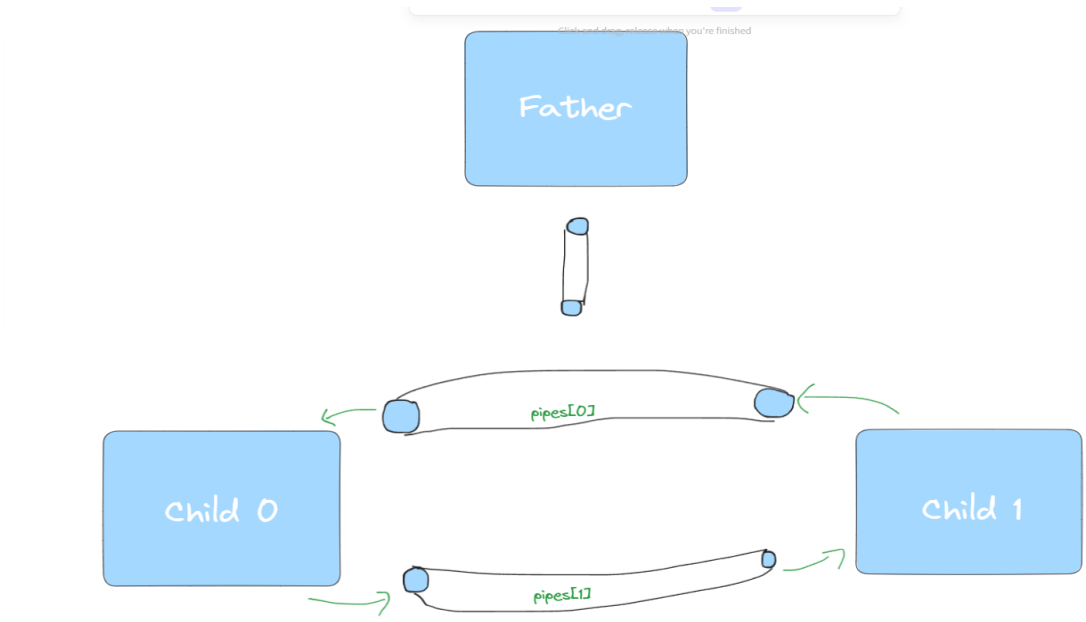
Dans ce rapport, nous allons vous présenter en détail le cheminement chronologique que nous avons suivi ainsi que les procédures que nous avons mises en place afin de vous proposer un dictionnaire consultable et entreposable.

Analyse du problème posé:

Vue globale :

Primo, l’utilisateur va renseigner comme argument de commande le nombre de processus , ensuite le père va s’occuper de la création des N noeuds fils , sachant que le seul moyen de communication entre les processus vu en cours et décrit en énoncé c’est les tubes , on doit en créer N tubes pour la communication entre les processus fils et le tube pour que tous les processus fils communiquent avec le controller , et un tube entre le père et les fils qui s'appellera mainPipe , cela est ainsi décrit dans notre schéma ci dessous.

Schéma 1



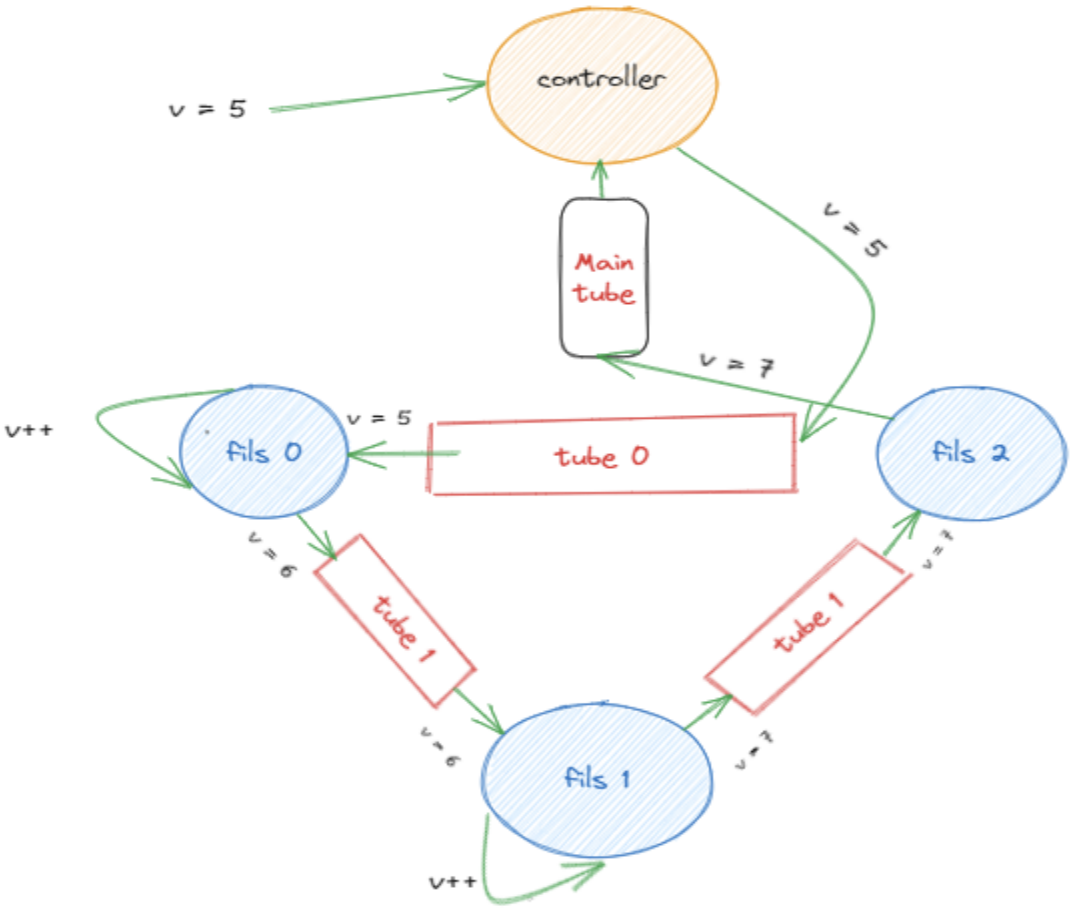
Initialisation des processus et de la communication:

Pour commencer, nous avons décidé de créer 3 processus qui incrémentent et qui communiquent au fils suivant la valeur donnée par le processus père.

Comment faire en sorte que les processus communiquent entre eux ?

Pour débuter, nous avons eu une approche qui consiste à enregistrer chacun des pid dans un tableau dans le main. Puis nous construisons le nombre de tubes adéquats.
Le père écrit un entier grâce à la fonction write() dans le tube précédant le fils 0, ferme tous les tubes créés sauf l'extrémité du tube qui communique avec le dernier fils, puis attend que le dernier fils lui écrive une valeur.
La fonction read() va permettre au processu fils 0 d'attendre la valeur donnée par le processus père.
Une fois celle-ci reçue, le fils incrémente la valeur puis l'envoie dans le tube le reliant au fils suivant. Nous procédons de manière itérative jusqu'au dernier fils. Nous nous sommes bien sûr assurés que chaque fils fermait tous les tubes une fois qu'il avait terminé sa tâche. Enfin, le dernier fils envoie dans le tube (="Main" tube) communiquant avec le père.

Schéma 2:



Problèmes rencontrés lors de la conception de la communication

Lors de la première création des processus et des tubes, nous avions détaillé le comportement du processus père dans la partie “default” du switch case du fork(). Cependant après plusieurs réécritures du programme, d’un débuggage important et de la lecture approfondie du professeur intervenant, nous avions une erreur qui persistait: le second processus fils n’arrivait pas à lire dans le tube qui le reliait au premier malgré une fermeture de tous les tubes après la tâche du premier fils terminée. Nous avons décidé de déplacer le comportement du père en dehors et à la suite du switch case mais dans la boucle for qui crée les processus fils. Cela a permis de résoudre ce problème et de passer à l’étape suivante.

Familiarisation avec la table:

Nous avons procédé à différents tests sur le fichier table.c donné en annexe.

Quelles informations pouvons-nous tirer de nos tests ?

Premièrement, la fonction store() permet de stocker une chaîne de caractères dans une liste chaînée à l’indice entré en paramètre de la fonction. Cela nous permet de déduire que chaque processus fils aura sa propre liste chaînée. Il faudra ainsi initialiser la liste chaînée lors de la création du processus fils.
Ensuite, la fonction lookup() donne lieu au renvoi de la chaîne à l’indice, entré en paramètre dans la fonction, de la liste chaînée stockée précédemment. Cependant, lors de nos tests, nous nous sommes aperçus que si la liste chaînée ne comprenait pas l’indice, la fonction faisait planter le reste du programme, il nous était impossible d’afficher le reste des fonctions.
Cela pourrait nous poser problème dans la suite de l’application car nous voudrions certainement pouvoir interagir avec le controller bien qu’il n’ait rien trouvé dans les listes chaînées de chaque processus fils.
Pour finir, la fonction display() concourt à afficher la liste chaînée. La problématique que nous pourrions en tirer est, comment souhaitons nous transmettre les chaînes de caractères au controller lorsque celui procèdera à la commande “dump”. Allons nous stocker toutes les listes chaînées renvoyées par la fonction display() de chaque node dans une seule et unique liste, lors de l'exécution du “dump”, puis la transmettre au controller par l’intermédiaire du “Main” tube ? Ou allons-nous envoyer chaque chaîne de caractère une à une au controller, tout en faisant une boucle au sein des processus pour respecter l’ordre des indices ? Nous répondrons à chacune des problématiques dans les parties suivantes.

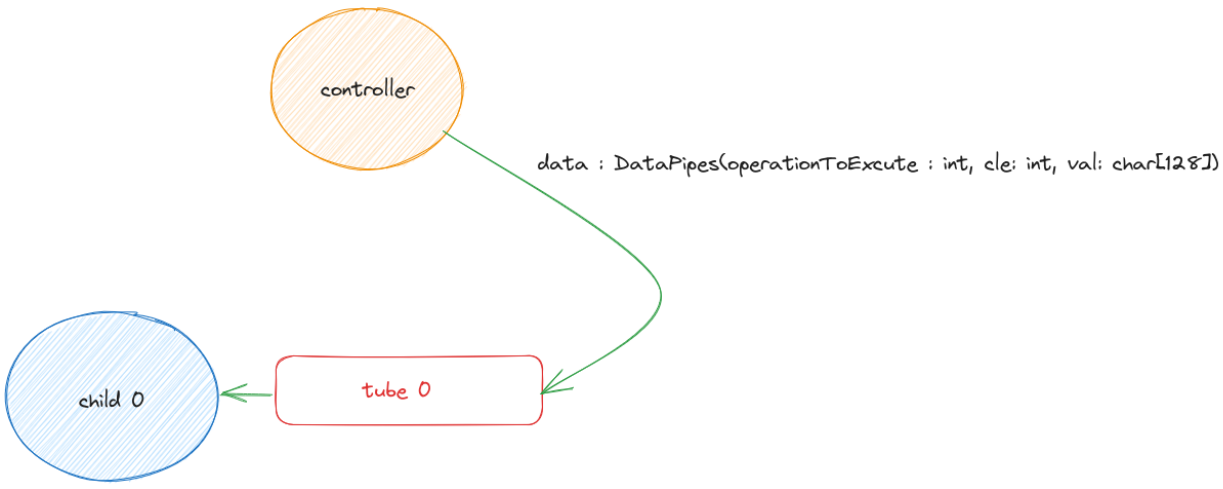
Réflexion sur l’assemblage des fonctionnalités

Après une longue introspection sur notre manière de combiner les différents aspects de notre application, plusieurs questions se présentent.

Qu’est qui sera transmis dans les tubes qui relient les processus fils entre eux ?

D’après l’énoncé et le schéma, on voit bien que le processus controller communique avec les processus fils il doit passer par un tube donc une seule donnée peut passer à la fois. Ainsi, quel sera le format de la donnée transmise par le controller ? Pour une commande set par exemple, on pourrait faire part de l’identifiant de l’opération à exécuter puis envoyer la clé et enfin la chaîne de caractère à stocker, le tout dans une chaîne de caractère qui sera décodée par chaque fils pour déterminer leur tâche. Cependant, nous avons pris une approche différente, nous avons décidé d’envoyer une donnée de type DataPipes qui stockera le code de la commande, la clé dans deux entiers distincts et la valeur dans une chaîne de 128 caractères comme indiqué dans l'énoncé. Nous assurerons, dans ce cas, la communication entre les processus par une seule donnée ainsi qu’un traitement simplifié lors de l'exécution du fils. Ceci est illustré de manière succincte dans le schéma ci-dessous.

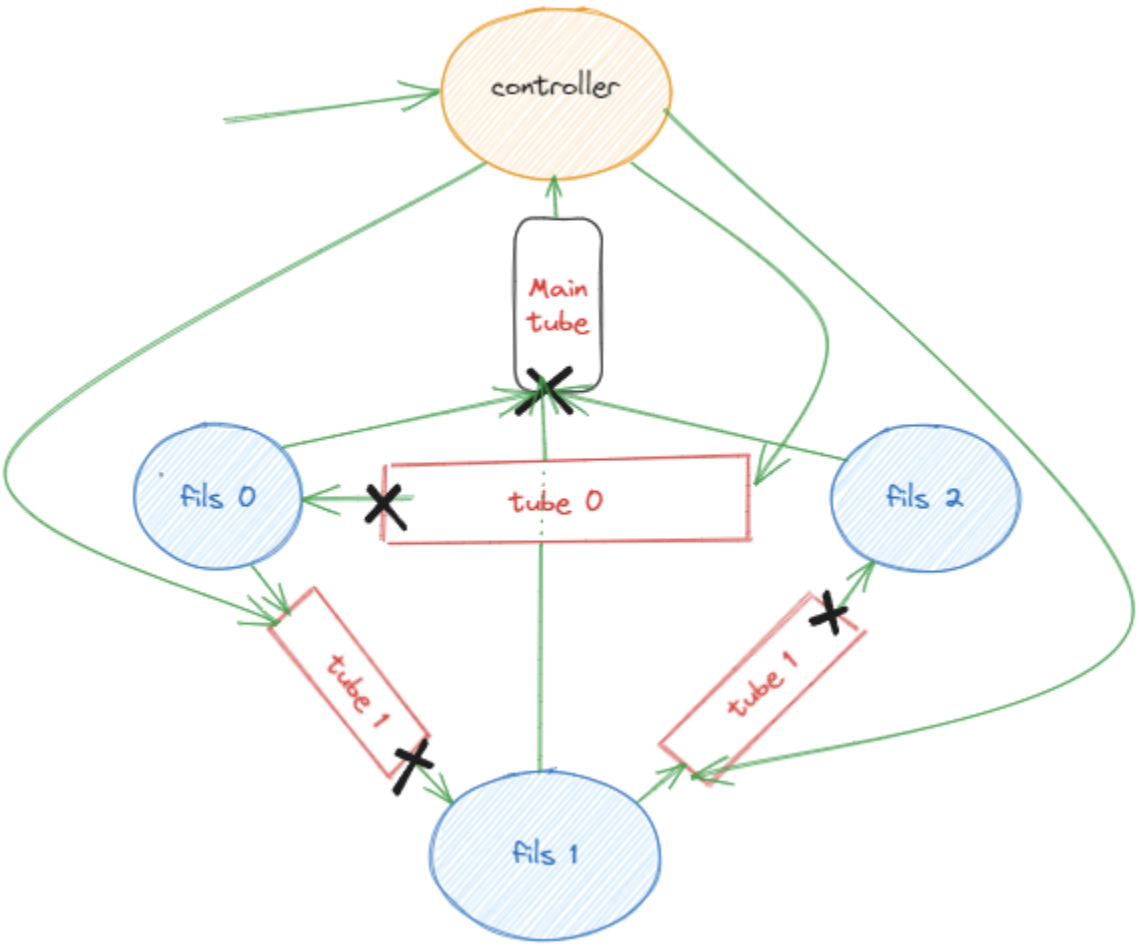
Shéma 3:



Comment faire en sorte que l'on puisse enchaîner les exécutions de commandes auprès du controller ?

Lors de la première étape d'initialisation, nous faisions le tour des processus fils puis nous revenions au controller qui affichait la valeur qui avait été incrémentée (cf: Schéma 1). Les extrémités des tubes étaient toutes fermées lorsque les processus avaient terminé leurs tâches pour ne pas avoir de zombies. Seulement, il nous est impossible de rouvrir les tubes, nous serions obligés de recommencer le programme et donc le stockage des données serait improbable. Nous allons donc devoir faire en sorte de fermer seulement le tubes non utilisés pour chaque fils pour que nous puissions encore communiquer avec les fils et continuer de leur envoyer des commandes. Par exemple, dans le schéma (=Schéma 4) ci-dessous nous devrions fermer les extrémités indiquées d'une croix si l'on se place au niveau du controller.

Schéma 4:

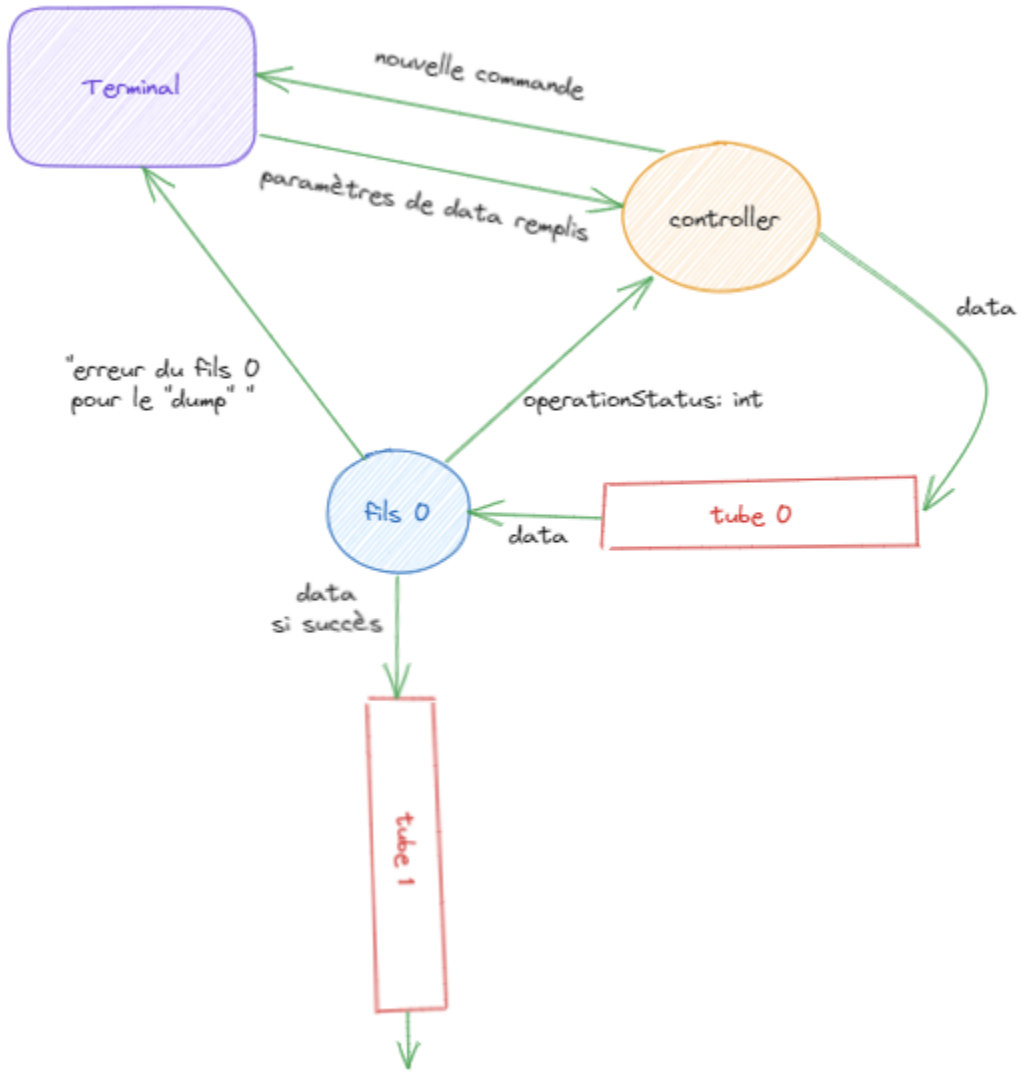


Nous avons pu remarquer, en l'occurrence, lorsque nous envoyions une commande au fils que le père nous demandait directement une commande sans attendre que le fils ait fini sa mission ou encore plus particulièrement lors de la commande "set" ou il ne donnait plus la main à l'utilisateur. Il a fallu donc que le père reste en lecture pour attendre et passer à la commande suivante, ce qui a résolu une partie conséquente des problèmes de synchronisation que nous avions rencontré jusqu'à présent.

Comment gérer les messages d’erreurs ou de fin d'exécution des fils ?

Cette problématique est liée à la problématique de synchronisation des processus. Lors de chaque envoi de commande aux processus fils, nous souhaiterions évidemment savoir si la tâche a été effectuée ou non. L’utilisation d’affichage de message pendant chaque différente réalisation d’un processus fils en particulier peut-être une solution avec l’emploi de la fonction printf(). Toutefois, au fur et à mesure que nous implémentions une nouvelle commande, le débogage devint fastidieux. De cette complication nous est venu l’idée d’ajouter une valeur qui varie suivant le statut du processus fils. Cet entier,communiqué directement au père à la fin de l'exécution du fils, permet de dialoguer avec le père de manière “invisible” tout en ayant une idée du statut du fils. Nous avons, néanmoins, gardé une partie de l’affichage des fils dans lorsqu’une erreur naissait. Nous l'illustrons dans le cas d’un “dump” dans le schéma 5.

Schéma 5:



Résolution de problèmes mineurs

Nous avons pu voir précédemment que la fonction lookup() fournie en annexe pouvait avoir des effets non escomptés, mais potentiellement utiles.

Que faire dans le cas d'une recherche d'une valeur qui n'existe pas ?

Lors de nos tests de la table, la fonction lookup() nous produisait une erreur en cas d'une valeur non trouvée et faisait planter le programme. Il fut donc nécessaire de remédier à cet ennui dans la commande "lookup". Au lieu d'envoyer "NULL" au père, ce qui le faisait crasher, nous envoyons une chaîne de caractères définie dans le fils qui serait envoyé au père qui l'affiche sur le terminal.

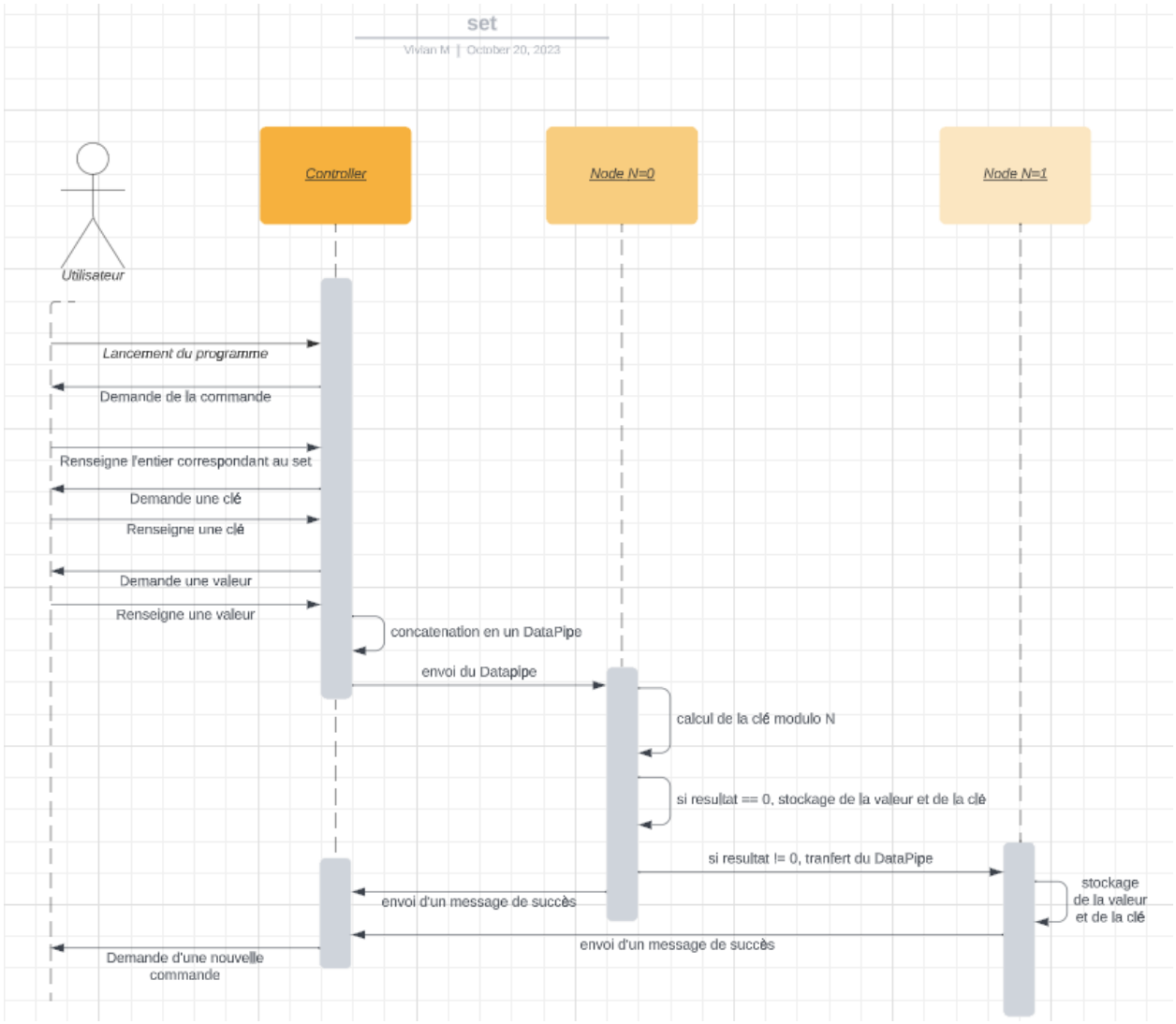
Comment éviter d'inclure une valeur déjà existante dans le dictionnaire lors d'un set ?

Durant nos implémentations de la commande set, une erreur nous est remontée suite à une valeur que nous avions renseigné auparavant qui se retrouve écrasée par une nouvelle valeur si nous lui avions attribué la même clé. Nous avons donc décidé de séparer la commande en deux. La première partie étant la recherche de la clé dans la liste chaînée du fils censé contenir la clé grâce à la fonction lookup(). Puis, suivant le résultat de cette dernière, nous redemandons à l'utilisateur de fournir une nouvelle clé ou nous stockons la valeur dans la liste chaînée.

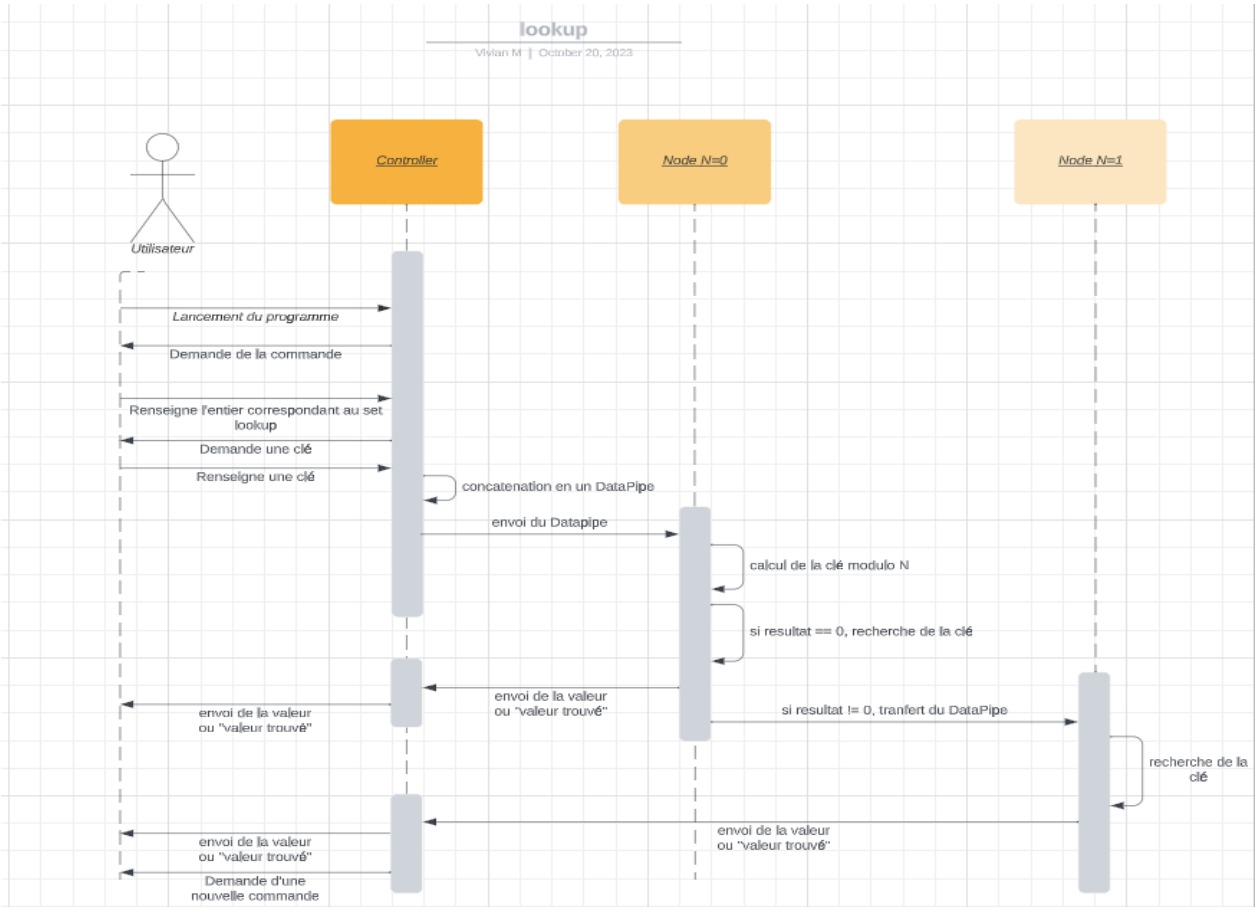
Diagramme de processus

Nous allons, dans cette partie, détailler chaque commande à l'aide de diagrammes de séquence.

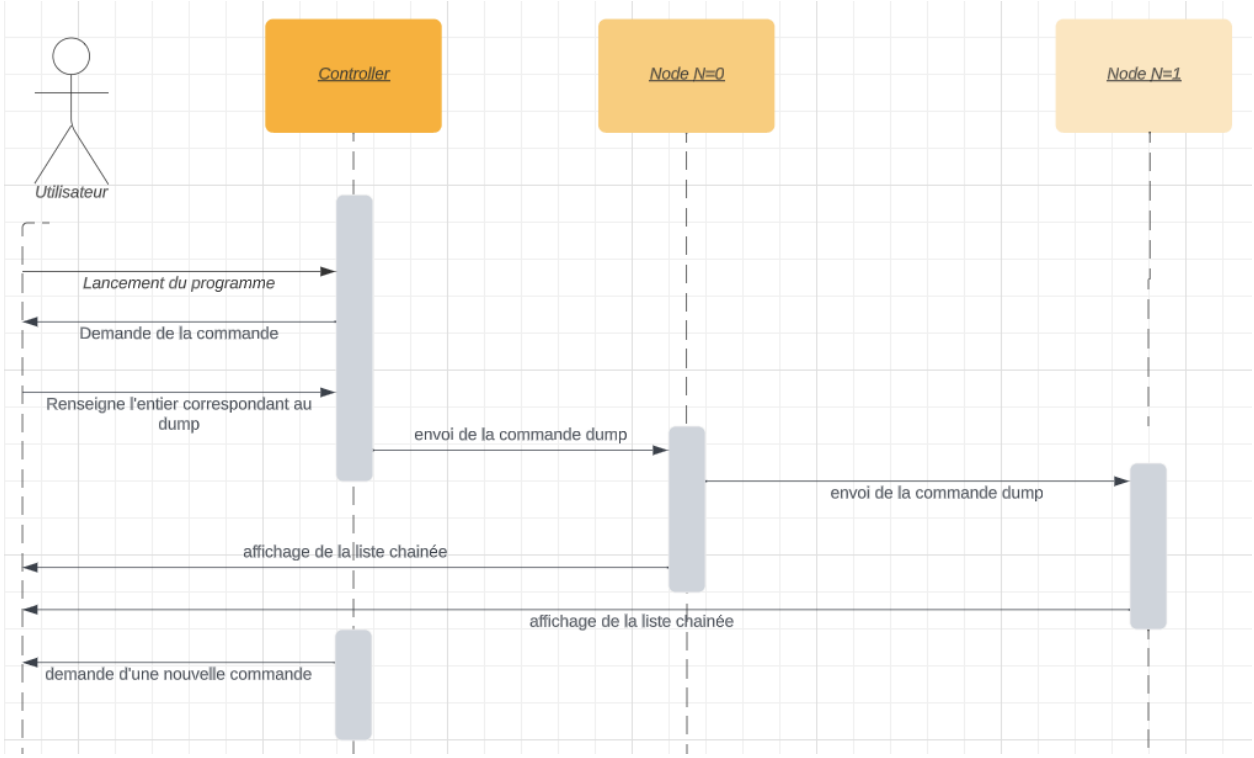
La commande SET



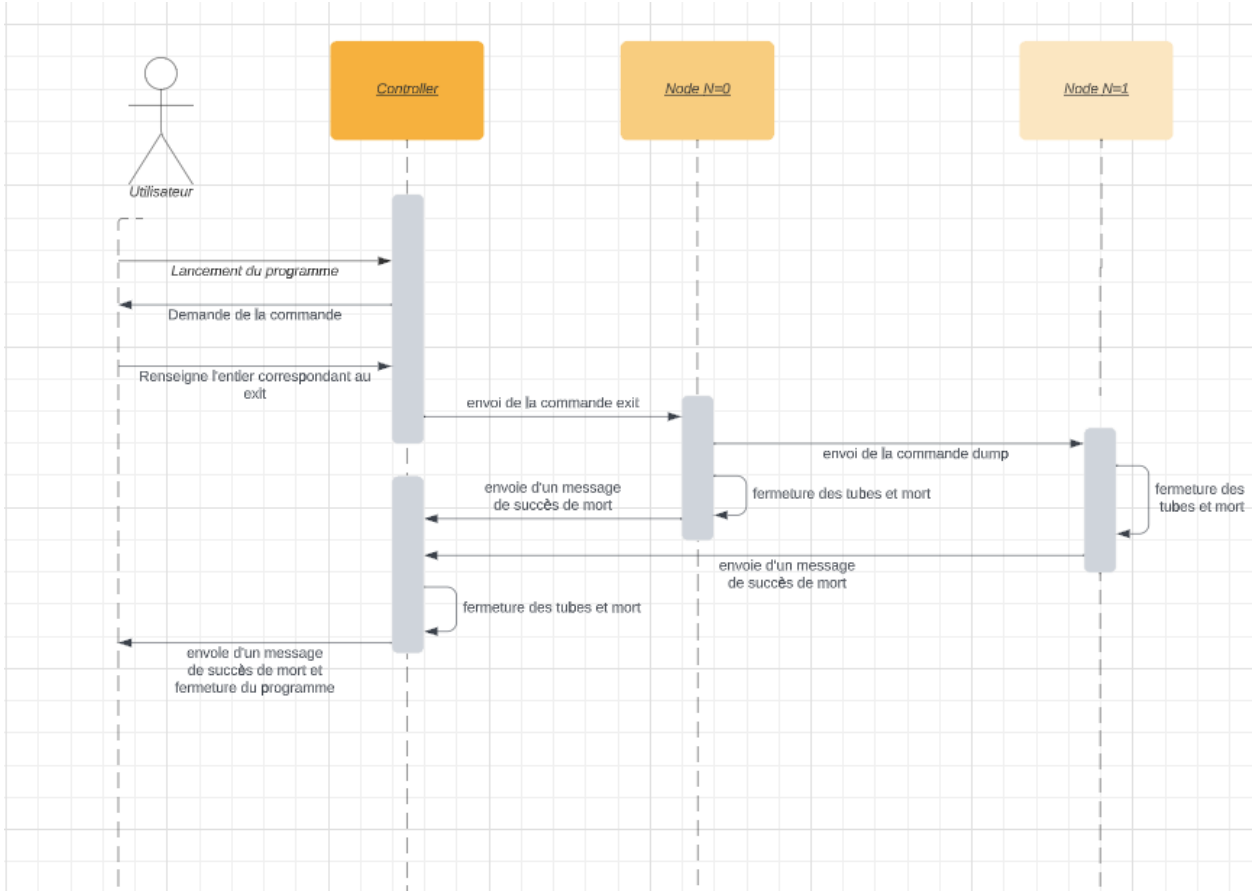
La commande lookup



La commande dump



La commande exit



Procédures d'exécution

Commandes à exécuter:

- make
- ./projetSys nombreDeProcessus
- make clean (supprime tous les fichiers.o)

Améliorations possibles

- La fonction node n’a pas été implémentée, nous avons gardé les fonctionnalités de processus fils dans le main() dû à la difficulté de déplacer les comportements dans cette fonction.
- Commande Dump: chaque node affiche sa liste chaînée sans pour autant prendre l'ordre des alphabétiques.