Introduction au systèmes

Begimay KONUSHBAEVA
Joël KONGOLO BEYA

Introduction

Nous avons pour projet de réaliser une application C de communication entre différents processus. Les processus vont être séparés en 2 catégories : le père et un nombre de fils indéterminé. Le processus père communique avec des processus fils. Chaque communication est réalisée à travers un tube. Chaque fils va stocker un dictionnaire de données, qui est formé via à une association clé-valeur et il est stocké sous forme de liste chaînée.

Objectif

Programmer une application qui permet de stocker et de consulter un dictionnaire des données

Grandes étapes

Partie I: Gérer la création des processus

Nous avons créé N nombres de processus fils a partir d'un processus père commun en utilisant une boucle avec la fonction fork.

Cette application fonctionne de la façon suivante :

1) On commence la boucle for à 0 jusqu'à N. À l'intérieur de la boucle on crée le premier processus fils. Puis on vérifie que c'est bien le processus père qui va créer les fills suivant en comparant la valeur de retour de la fonction fork avec 0:
    - si la valeur de retour de la fonction fork() est égal à 0 nous sommes dans le processus fils
    - si la valeur de retour de la fonction fork() est différente de 0 nous sommes dans le processus père

Donc nous avons comparé la valeur de retour de la fonction fork() et 0. Si cette dernière est égale à 0 nous avons créé un processus fils. On le laisse mourir tout de suite à l'aide de la fonction exit()

2) Si la valeur de retour de notre fonction fork n'est pas égale à 0, le programme sort de la boucle if (c'est-à-dire dès que c'est le père qui exécute le code). Le père quant à lui dans la même boucle (rappel : de 0 à N) attends la mort de tous ses fils. Donc, le père exécute wait(NULL) N fois.

Après l'appel de la fonction fork() sera doublé par le fils que cette fonction nous a créé

Pour éviter la création des petit-fils (des processus fils créés par d'autres processus fils) il faut tuer le processus fils (dans notre cas juste après sa création).

Compilation de l'application:

Pour tester notre application::

1. On définit notre fonction créer_processus décrite ci-dessus
2. Le main de ce programme demande à l'utilisateur de saisir le nombre de processus voulu.
3. Pour chacun des processus créé notre programme affiche son numéro et son pid.

Nous pouvons ainsi voir que le PID de chaque processus suivant est toujours supérieur au PID du processus précédent.

Pour s'assurer que tous les processus fils sont bien morts et qu'il ne reste pas des processus "zombie" nous vérifions l'état de tous nos processus:

En exécutant l'option -l de la commande ps sur le terminal, nous voyons qu'effectivement on n'a qu'un seul processus à la fin.

Exemple de l'exécution

```
[begimaykonushbaeva@MBP-de-Begimay Projet % ./projet_0
 Entrez le N : 6
 On a créé un processus 1865
 On a créé un processus 1866
 On a créé un processus 1867
 On a créé un processus 1868
 On a créé un processus 1869
 On a créé un processus 1870
```

Partie II: Gérer la communication de processus

Conception de l'application

Utilisation de pipe

La fonction pipe fonctionne de la même façon que la fonction de l'ouverture des fichiers. Cette fonction prend en argument un tableau de deux entiers, chaque entier représentant l'extrémité de ce pipe (tube en français):

- 0 : extrémité de tube en lecture
- 1 : extrémité de tube en écriture

Dans la deuxième partie nous utiliserons le même principe décrit dans la partie I pour la création d'un nombre de processus indéfini.

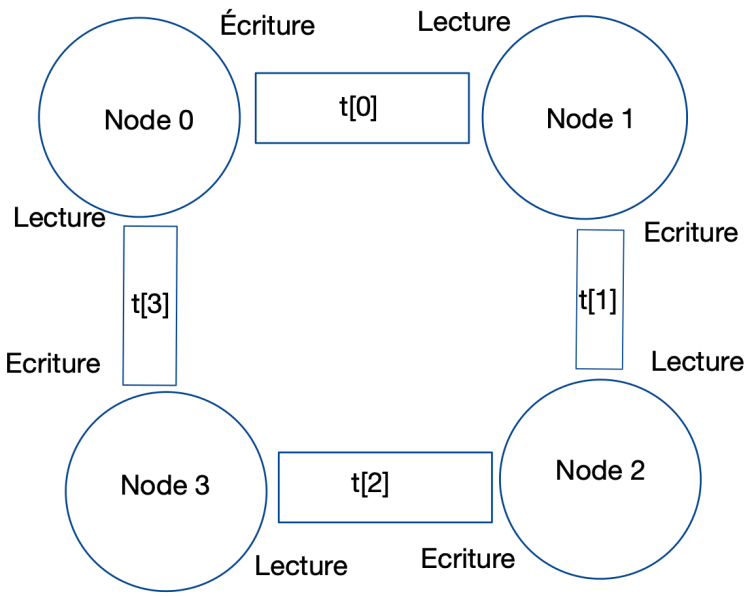
Pour faire communiquer tous les processus fils entre eux nous avons construit un anneau de processus avec un tube de plus pour le processus père.
Pour fabriquer une tels structure on passe par les étapes suivantes :

- Construction d'un anneau de processus fils
- La communication de cet anneau avec le processus père

Ces tubes communiquent à l'aide des fonctions read() et write().

Anneau des processus

Pour faire communiquer tous les processus créés nous avons fabriqué un anneau des processus



- Pour transmettre de l'information entre les processus on devrait avoir N pipes, qui correspond à N fils.
- Pour ouvrir un nombre indéfini des pipes on a proposé la création d'un tableau à deux dimensions : le nombre des nodes souhaités et 2 extrémités de chaque tube.
- Chaque processus fils (node dans la suite) récupère une donnée du node précédent et transmet cette données dans le node suivant.

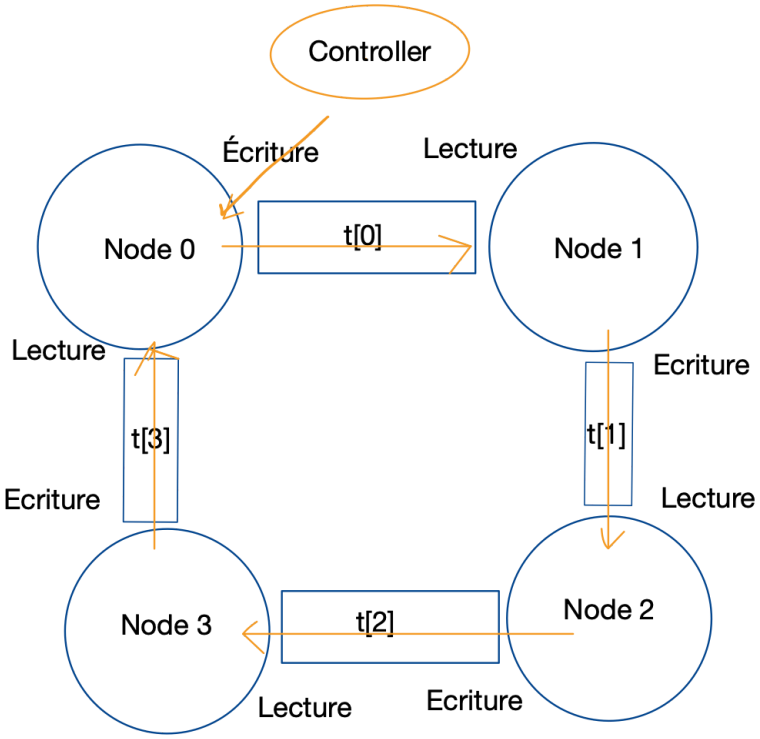
Lecture et écriture: fonctions read et write

La fonction read : prend en paramètre un descripteur de fichier, la donnée qu'il faut lire de ce fichier et la taille de cette donnée en bits. (exemple: pour une donnée de type int c'est sizeof(int), pour une donnée de type char c'est sizeof(char), c'est-à-dire que l'on devrait connaître le type et la taille de la donnée a lire.

Pour envoyer certaines requêtes dans les nodes nous devrons envoyer plusieurs données à la fois. Comme décrit ci dessus on renvoie une donnée et pour transmettre plusieurs données, on écrit dans le tube une donnée à la fois.

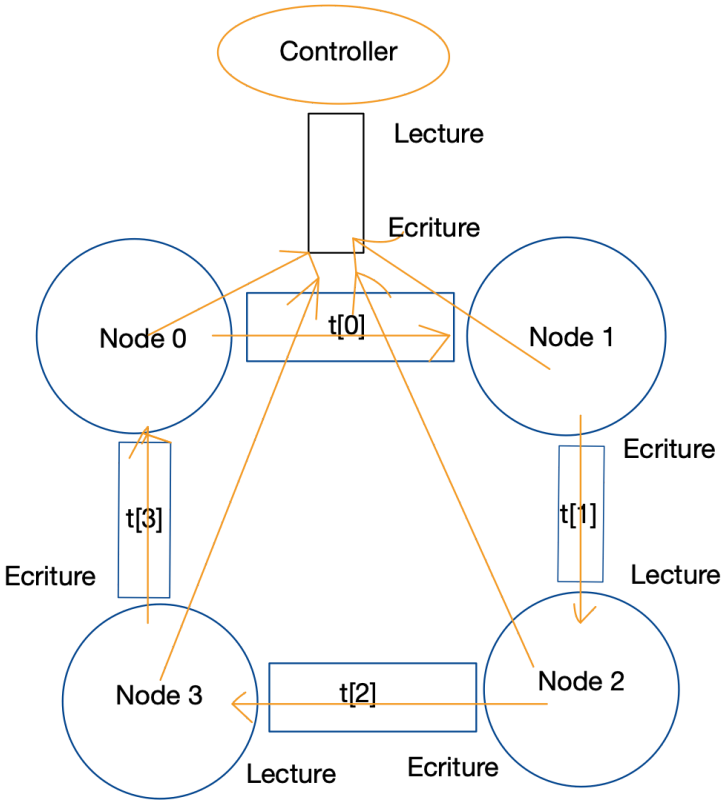
Cela vient du fait que les fonctions read et write utilisent la technique FIFO : first in first out.

La communication de l'anneau avec le controller(processus père)



Pour pouvoir transmettre de l'information externe dans cet anneau il faut qu'il y ait un controller qui (suivant les consignes de la planche de TP) va transmettre les données dans le premier node : Node 0.
Le "controller" se trouve dans le processus père:
Après avoir créé un anneau des nodes qui font circuler les données entre eux, où chaque node est un processus fils, le programme revient dans le processus père ("controller" dans la suite). Nous avons proposé une méthode de communication entre le controller et l'anneau de nodes par implémentation de la fonction write bloquant dans le processus père (controller). Cette fonction nous permet d'exécuter les instructions qui se trouvent dans la node déjà créé (dans notre cas c'est le node 0) avant d'exécuter les instructions suivantes.

De plus, suivant les consignes de la planche de TP, tous les nodes peuvent écrire dans, d'où le controller peut lire.



La réalisation de l'application

1. La première étape comprend la création de N processus node. Pour faire cela on utilise la methode decrite dans la partie I
2. Pour effectuer la communication entre tous les nodes on ouvre N pipes, comme décrit dans l'anneau des processus.
3. Puis on définit les tubes de lecture et l'écriture. Chaque node il lit dans le tube i-1 a part le node 0, qui quant à lui lit dans le tube du dernier node c'est-à-dire dans le tube N-1.
4. Une fois dans un processus fils on ferme tous les fichiers dont on n'aura pas besoin, donc tous a part le tube ou on écrit/lit.
5. Après avoir lu la donnée le node i l'envoie au node i+1 par le tube i (par exemple pour 4 nodes : le node 0 lit dans le node 3 et envoie dans le node 1 par le tube 0, le node 1 lit dans le node 0 et envoie dans le node 2 par le tube 1, le node 2 lit dans le node 1 et envoie au node 3 par le tube 2, le node 3 lit dans le node 2 et envoie dans le node 0 par le tube 3.
6. Dans le processus père on demande à l'utilisateur de saisir une commande (un entier qui sera transmis au processus node suivants)
7. Avec un write bloquant on écrit cette commande dans le node 1.
8. Après l'exécution de la fonction controller dans le main le processus attends la mort de tous ces processus fils.

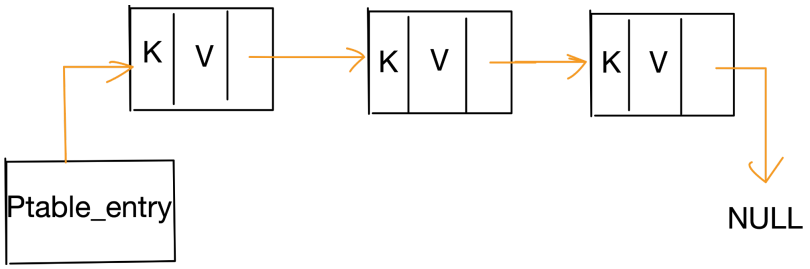
Exemple d'exécution

```
begimaykonushbaeva@MBP-de-Begimay Projet % ./projet_1 5
On a créé un processus Node 0, avec le pid 2034
On a créé un processus Node 1, avec le pid 2035
On a créé un processus Node 2, avec le pid 2036
On a créé un processus Node 3, avec le pid 2037
On a créé un processus Node 4, avec le pid 2038
Saisir la commande :
1
La valeur 1 lue par le Node ,avec le pid 1 : 2035
La valeur ecrite par le fils n°1 ,avec le pid 2035 : 2
La valeur 2 lue par le Node ,avec le pid 2 : 2036
La valeur ecrite par le fils n°2 ,avec le pid 2036 : 3
La valeur 3 lue par le Node ,avec le pid 3 : 2037
La valeur ecrite par le fils n°3 ,avec le pid 2037 : 4
La valeur 4 lue par le Node ,avec le pid 4 : 2038
La valeur ecrite par le fils n°4 ,avec le pid 2038 : 5
La valeur 5 lue par le Node ,avec le pid 0 : 2034
La valeur ecrite par le fils n°0 ,avec le pid 2034 : 6
La valeur revenue au pere : 6
```

Pour s'assurer que les nodes communiquent bien entre eux nous avons incrémenté la valeur de la variable de test commande de 1 dans chaque processus node.

Partie III : Se familiariser avec la collection associative.

Listes chaînée



Chaque élément de la liste chaînée fournie dans l'annexe de la planche de ce TP comprends:

- une clef (k sur le schéma ci-dessous, key dans la structure décrite dans l'annexe) est un entier
- une valeur (v sur le schéma, val dans le structure décrite) est une chaîne de caractères
- un pointeur vers l'élément suivant

Ptable_entry est le pointeur vers le premier élément de cette liste (tête le de la liste).

Les fonctions

Nous avons 3 fonctions fournies en annexe de la planche du TP:

store : la fonction pour stocker une donnée dans une liste table avec une clé k est une valeur v passés en argument de la fonction.

lookup : la fonction pour chercher la valeur de clé k passé en paramètre.

display : la fonction pour afficher tous les éléments de la liste (la clé et la valeur)

Exemple d'exécution

```
begimaykonushbaeva@MBP-de-Begimay Projet % ./testTable
saisir le nombre de valeur dans la liste : 2
Saisir la valeur n°1
Un
Saisir la valeur n°2
Deux
------------------------AFICHAGE DE LA TABLE-----------------------
2 : Deux
1 : Un
------------------------RECHERCHE D'UNE VALEUR---------------------
saisir la clé : 2
la valeur trouvée est : Deux
```

Partie IV: Assemblage des fonctionnalités

Pour réaliser ce projet on utilisera :

- le principe de la création de N processus décrit dans la partie I.
- l'anneau de ces N processus et leur communication avec le processus père (controller) décrit dans la partie II.
- Un dictionnaire sous forme d'une liste chaînée décrit dans la partie III.

Conception de l'application

La réalisation de cette partie comprend : le programme principal et la fonction controller.

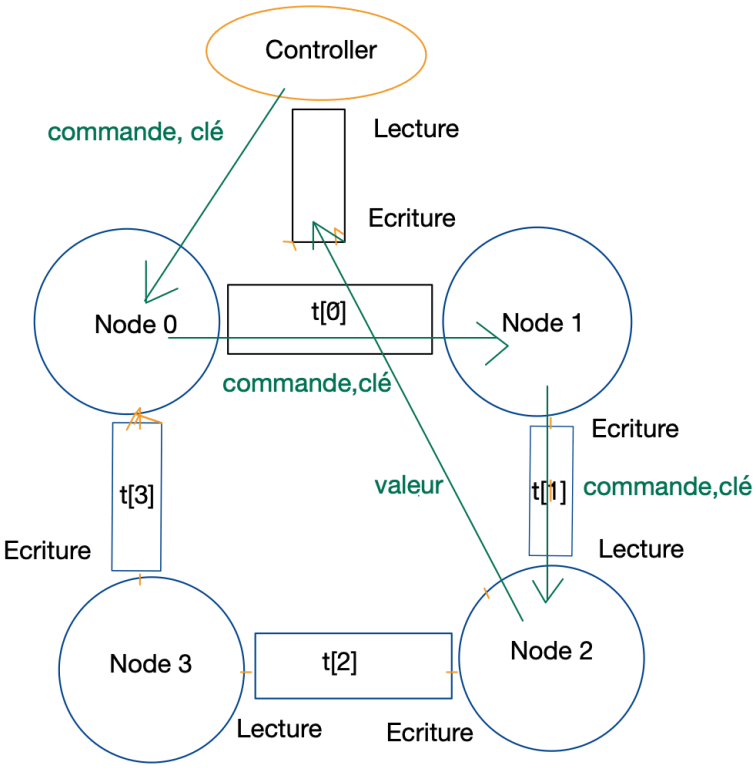
Le programme principal:

Le programme principal récupère N - le nombre de nodes de l'entrée standard (passé en argument avec l'exécutable. Dans les cas d'erreurs (mauvais nombre d'arguments ou erreurs de lecture d'un entier) le programme fait l'appel à la fonction perror. Cette fonction mets dans le tableau errno, qui a été automatiquement initialisé avec l'appel à la fonction fork une chaine de caractère que l'on passe en paramètre en appel de cette fonction.

Si il n'y a pas d'erreur dans les données saisies on définit un dictionnaire vide en lui allouant de l'espace en mémoire avec malloc. Puis avec ces 2 donnée : nombre des nodes voulu et le dictionnaire on fait l'appel à la fonction controller.

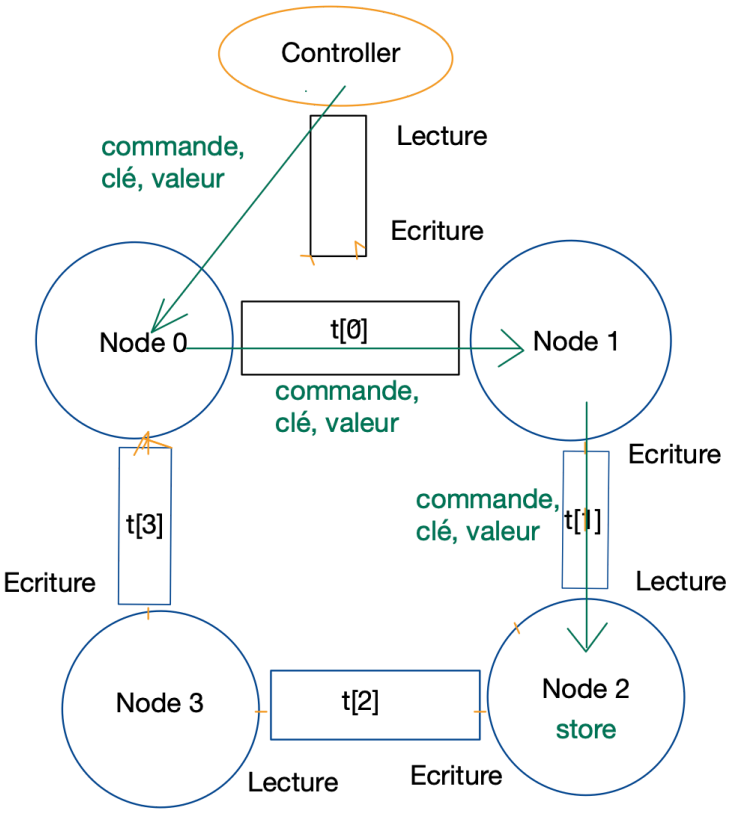
Exemples des commandes

La commande set



Si la commande est set, le controller envoie cette dans le node 0 ensuite dans le node 1 et ensuite. À gauche voici un exemple avec 4 nodes est une clé 10. (On stock la cette clé avec la valeur saisie dans le processus 10%4 = 2

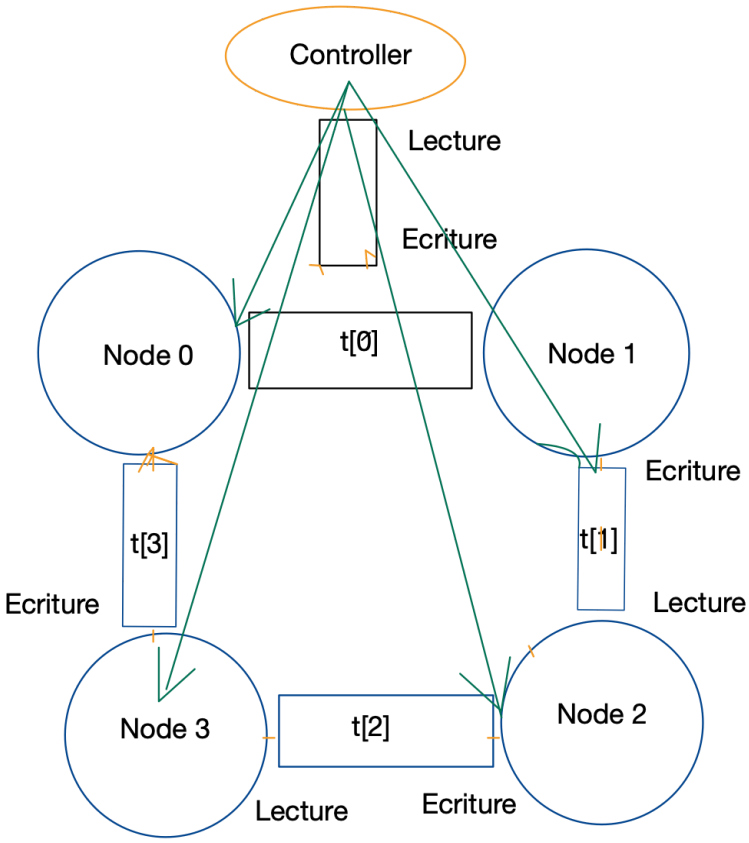
Commande lookup



À gauche voici un exemple de recherche de la valeur que nous avons mis dans le dictionnaire de node 2 dans l'exemple precedent. Le controller envoie la commande est la clé au node 0. Dans le tube de controller le node 2 renvoie la valeur retrouvée au controller

Les commandes exit et dump

Les commandes exit et dump sont envoyé a tous les nodes simultanément , comme illustré à droite



La fonction controller

Cette fonction prend en argument le dictionnaire table et le nombre des nodes à créer. Puis en fonction de la commande elle définit le comportement des processus node et le controller.

- Pour transmettre et manipuler les données on déclare un entier : commande la commande qui sera saisi par l'utilisateur, un entier : k -clé que l'on demandera à l'utilisateur pour les commande 1(set) et la commande 2(lookup)
- On utilisera un tableau des tubes de taille N comme décrit dans la partie 2. Donc on crée un tableau et on ouvre tous les tubes de ce dernier.
- Pour créer et manipuler les nodes (les processus fils) on crée les node utilisant le principe décrit dans la partie I (appel a la fonction fork N fois dans le processus père)
- On définit les tubes la lecture et de l'écriture comme décrit dans la partie II (le tube in - le tube en extrémité de lecture à l'indice N-1 pour le node 0 et à l'indice i-1 pour tous les autres nodes, le tube out - le tube en extrémité de l'écriture à l'indice i pour tous les nodes). On ferme tous les tubes dont on n'aura pas besoin.
- Une fois rentré dans le node on lit la commande de tube in.
- Après que la commande a été lue, on définit le comportement pour chaque commande: (0 - la commande exit, 1 - la commande set, 2 - la commande lookup, 3

- la commande dump). Les comportement de chaque commande dans les nodes seraient décrits dans la suite.
- Dans le controller on demande à l'utilisateur de saisir la commande jusqu'à ce que cette dernière soit 0.

Le comportement de controller en fonction de la commande:

- 0 (la commande exit) : on envoie cette commande simultanément dans tous les tubes de 0 à N. Pour implémenter ça on écrit la commande dans le tube i en extrémité de l'écriture dans une boucle de 0 à N.
- 1 (la commande set) : après avoir reçu cette commande on demande à l'utilisateur de saisir la clé et la valeur que l'on va mettre dans le dictionnaire. Puis on écrit la commande, la clé et la valeur dans le tube 0 un par un
- 2 (la commande lookup) : après avoir reçu cette commande on demande à l'utilisateur de saisir la clé pour rechercher la sa valeur. Puis on écrit la commande et ensuite la clé un par un.
- 3 (la commande dump) : on envoie cette commande dans tous les tubes simultanément. Suivant la méthode décrite pour la commande 0 (exit) on envoie la commande dans le tube i dans une boucle de 0 à N.

Le comportement de nodes en fonction de la commande :

- 0 (la commande exit) : cette commande est transmise par le controller dans tous les tubes simultanément. Donc tous les node reçoivent cette commande de tube et exécutent exit(0) (la mort de tous les nodes)

  1 (la commande set) : cette commande est transmise par le controller dans le premier node. Donc on la lit dans le tube in. Une fois dans le bon processus (le node est le reste de la division de la clé par le nombre de nodes), si ce n'est pas le bon processus le programme transmet la commande, la clé et la valeur au node suivant un par un.

- 2 (la commande lookup) : cette commande est transmise par le controller par le tube 0 au node 0. De la même façon que la commande set, la commande lookup cherche le bon processus. Si le bon processus est trouvé on recherche la valeur avec la clé dans le dictionnaire de ce node.
- 3 (la commande dump) : de la même façon que la commande exit cette commande est transmise au tous les nodes simultanément. Une fois la commande reçue, si le dictionnaire de ce node n'est pas vide, on affiche le pid et le dictionnaire de ce node.

Exemple d'exécution

```
[begimaykonushbaeva@MBP-de-Begimay Projet % ./projet 5
Saisir commande (0 = exit, 1 = set, 2 = lookup, 3 = dump): 1
Saisir la cle (decimal number): 12
Saisir la valeur (chaine de caracteres, max 128 chars) : Douze
Saisir commande (0 = exit, 1 = set, 2 = lookup, 3 = dump): 1
Saisir la cle (decimal number): 16
Saisir la valeur (chaine de caracteres, max 128 chars) : Seize
Saisir commande (0 = exit, 1 = set, 2 = lookup, 3 = dump): 2
Saisir la cle (decimal number): 12
Valeur trouvée = Douze
Saisir commande (0 = exit, 1 = set, 2 = lookup, 3 = dump): 2
Saisir la cle (decimal number): 13
Valeur trouvée = (null)
Saisir commande (0 = exit, 1 = set, 2 = lookup, 3 = dump): 3
process : 2114
12 : Douze
0 : (null)
process : 2113
16 : Seize
Saisir commande (0 = exit, 1 = set, 2 = lookup, 3 = dump): 0
bye bye !
```

Makefile

Il contient tous les exécutables utiles à la compilation de chaque partie de cette application.

Initialement l'application est organisée de la manière suivante:

- un répertoire exec : Qui contiendra tous les exécutables
- un répertoire obj : Qui contiendra tous les fichiers objets
- un répertoire src : Contient tous les fichiers .c et les fichiers headers.

À la compilation, les répertoires exec et obj contiendront respectivement tous les exécutables générés et les fichiers objets.

Les exécutable produites par le makefile

- Partie I : création des processus : projet_1
- Partie II: communication des processus : projet_0
- Partie III : se familiariser avec la collection associative : testTable
- Partie IV : Assemblage des fonctionnalités : projet