2023/2024

Application qui permet de stocker et de consulter un dictionnaire de données

Rapport de projet

Systèmes & Réseaux L3 MIAGE

Réalisé par :

- OURZIK Jugurta
- MIEL Nils

Descriptif du projet

Le but de ce projet est de programmer une application qui permet de stocker et de consulter un dictionnaire de données. Les données seront de type chaîne de caractères.

Le dictionnaire sera une collection associative, c'est-à-dire que les données seront stockées grâce à une association clé-valeur, la clé de stockage étant un entier. Plus précisément, nous utiliserons une liste chaînée de couples clé-valeur.

Le dictionnaire sera réparti entre N processus node, N ≥ 2.

Ce nombre de processus sera choisi au lancement du programme, passé en argument à la commande. Chaque processus node numéro i prend en charge les valeurs de clés k = i, i+N, i+2N,… Ces processus seront les processus fils d'un processus contrôleur assurant l'interface avec l'utilisateur.

Ce controller donnera la possibilité à l'utilisateur de saisir des commandes :

- "set" (code commande 1),
- "lookup" (code commande 2),
- "dump" (code commande 3),
- "exit" (code commande 0).

Structures et types de données:

Les données concernées par le stockage sont de type **“chaîne de caractères”** et sont associées à une clé **k** de type “**entier”**.

Ces données seront stockées dans une liste chaînée.

Étape 1: Analyse

Les données qui seront transmises sont :

- La commande de la requête souhaitée + la clé k.
- La commande de la requête souhaitée + la clé k + la valeur de la chaîne à stocker.
- La commande de la requête souhaitée.

Protocole de communication:

Pour propager l'information, nous avons décidé d'envoyer en plusieurs fois l'information, par exemple si on a besoin de communiquer une commande ainsi que la clé et la valeur, alors via un buffer nous allons envoyer d'abord la commande, puis la clé, et enfin la valeur. Même si la commande et la clé sont des int, nous préférons utiliser un buffer de type char* puisque la valeur sera de type char*. Cela ne pose pas de problème puisque l'on saura toujours ce qu'on attend (commande,clé ou valeur), autrement dit on connaît l'ordre des données transmises.

Par exemple, si on envoie la commande 0 (exit) , on sait qu'on ne va rien lire après et s'il s'agit de la commande 2 (lookup), dans ce cas on lit une deuxième fois pour extraire la clé du pipe.

Ce protocole permet de communiquer les processus et transmettre de l'information d'un processus à un autre.

1.La commande de la requête + la clé:

La commande est aussi un entier positif (0,1,2 ou 3) qui est saisie par l'utilisateur en premier qui sera vérifiée:

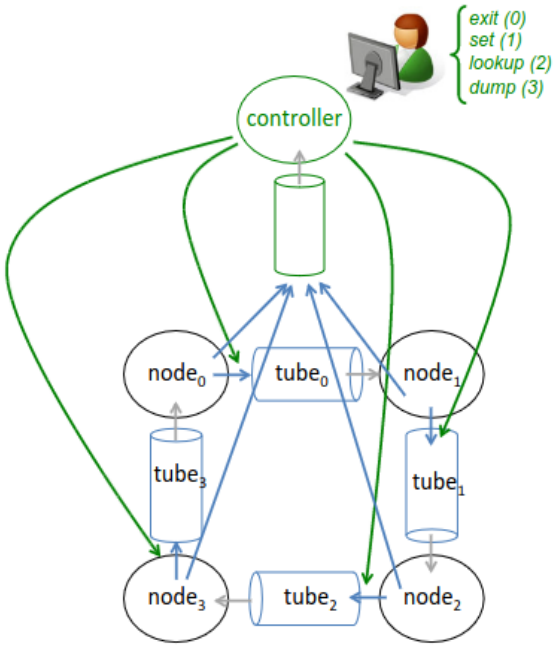
- "set" (code commande 1),
- "lookup" (code commande 2),
- "dump" (code commande 3),
- "exit" (code commande 0).

La clé k est un entier associé à la valeur "chaîne de caractères" ,est lue depuis l'entrée standard. L'utilisateur saisit sa clé qui ensuite sera vérifiée par le controller.

Elle sera transmise par le contrôleur assurant l'interface avec l'utilisateur au nœud 0 (premier processus) à travers le tube N, le nœud 0 lit la valeur de ce dernier.

Le processus 0 cherche la valeur associée à cette clé et remonte l'information au contrôleur si la recherche a réussie , sinon ce dernier la fera propager aux autres processus nœuds à travers le tube 1 puis le tube 2,....,tube x , jusqu'à ce que le destinataire effectif soit atteint.

Une fois le processus effectif atteint, il procédera également à la recherche de la valeur associée à la clé donnée, remonte l'information au contrôleur concernant la réussite ou pas de la recherche.



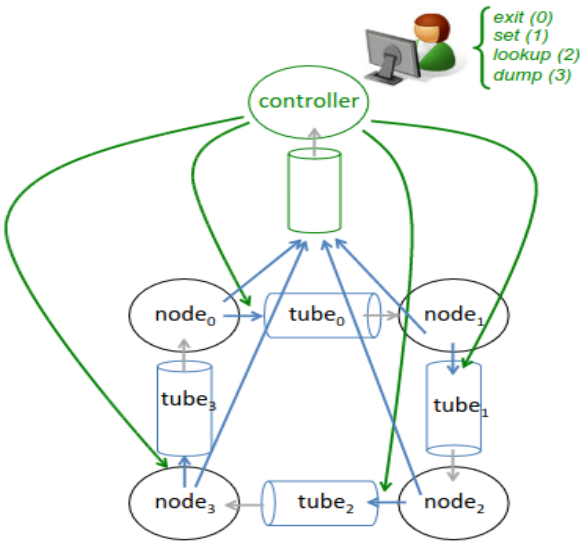
2. **La commande de la requête + la clé + la valeur:**

Ici il s'agit de trois données , la commande, la clé et la valeur. Les trois seront lues depuis l'entrée standard à la demande du contrôleur et saisies par l'utilisateur.Elles seront également vérifiées par le contrôleur avant de procéder au traitement de la requête.

Si l'une des données n'est pas valide l'utilisateur sera dans l'obligation de saisir à nouveau une entrée valide.

La clé et la valeur associée seront transmises par le contrôleur au premier processus (nœud 0) à travers le tube N.Celui-ci vérifie s'il gère la clé associée ou pas.Si le processus 0 gèrait la clé , il fera dans ce cas la sauvegarde de la donnée et informe le processus père, sinon les trois valeurs seront propagées dans l'anneau de processus, jusqu'à ce que le destinataire effectif soit atteint.

Dans les deux cas (réussite ou échec), le contrôleur s'il reçoit une information de la part de ses processus fils , affiche cette dernière et envoie un signal pour tuer ces fils (fin du cycle de la requête) et demande à nouveau à l'utilisateur une commande à traîter.



3.La clé: la commande dump, comment synchroniser les processus ? :

Pour synchroniser l'affichage des processus, nous allons tout simplement les faire communiquer entre eux. Tout d'abord le contrôleur enverra grâce à une boucle, la valeur de la commande à chaque pipe. Ensuite tous les nodes vont parcourir en même temps leur liste chainée associée, pour former une chaîne de caractères qui correspondra à l'affichage du contenu d'un processus demandé :

```
Processus numéro x :
    clé i : valeur i
    clé i suivante : valeur i suivante
    ....
```

Ensuite si le node n'est pas le numéro 0 , celui-ci attendra au niveau d'un read que le

node précédent lui envoie l'ordre d'afficher. À ce moment-là, le node courant affichera sur la sortie standard la chaîne de caractères qu'il a construite précédemment. Puis à l'aide d'un write, il enverra l'ordre au node suivant d'afficher. Dans le cas du node 0, celui-ci n'attendra pas d'ordre pour afficher, il le fera directement puis donnera l'ordre au processus 1 d'afficher. Cependant après cela le node 0 sera en attente de l'ordre d'afficher provenant du node N-1, une fois l'ordre reçu, il transmettra l'information au contrôleur que l'affichage des valeurs ait terminé, pour que celui-ci mis en attente puisse afficher de nouveau le menu principal.

Pour cela nous avons modifié la fonction display :

```
void display(PTable_entry table);=====> char* toString(PTable_entry table);
```

Désormais la fonction n'affiche plus sur l'entrée standard mais génère à la place une chaîne de caractères contenant l'affichage complet de la liste Chaînée.

Étape 2: Gestion et création des processus

Le nombre de processus à créer dépend du nombre choisi par l'utilisateur. Il sera récupéré depuis l'entrée standard par le contrôleur est stocké dans une variable N.

Pour créer **l'anneau de processus**, il faut **autant de tubes que de processus.**

Hiérarchie de processus

La création se fera à l'aide d'une boucle for allant de 0 à N exclut et en appelant la fonction système **fork()** . Chaque tour de boucle correspond à la création d'un processus. Ces derniers seront indexés avec le même indice à savoir la variable

d’itération i de la boucle.

Les processus sont créés au même niveau, autrement dit ils ne sont pas imbriqués.

Où sont stockés les pipes ?

La fonction **`pipe`** crée un canal, c’est-à-dire un canal d’E/S artificiel qu’un programme utilise pour transmettre des informations à d’autres programmes (ici les autres processus). Un canal ressemble à un fichier, car il a un pointeur de fichier, un descripteur de fichier ou les deux. De plus, il peut être lu à partir de ou écrit dans à l’aide des fonctions d’entrée et de sortie de la bibliothèque Standard. Toutefois, un canal ne représente pas un fichier ou un appareil spécifique. En effet, il représente un stockage temporaire en mémoire qui est **indépendant** de la mémoire propre au programme et qui est contrôlé entièrement par le système d’exploitation.

Il faut donc stocker les tubes dans une structure de données pour pouvoir fermer **à tout moment** les descripteurs de fichiers non nécessaires au fonctionnement du processus courant, Ici on a choisi de les stocker dans un **tableau 2D de taille N** dont l’indice de chaque case **i** représente le **tube** reliant le **processus i** au **processus i+1**.

On crée donc les N pipes et on initialise chaque pipe correspondant au processus courant d'indice i.

Pour les N listes chaînées,elles sont également initialisées à chaque tour de boucle et cela dans un tableau 1D où la case d’indice i représente la liste chaînée correspondant au processus i.

Que se passe t’il après la naissance d’un fils?

Après la naissance du fils, le processus père sortira du switch pour faire un nouveau parcours de la boucle, une fois la boucle terminée et tous les processus fils créés, le père rentrera dans la boucle du contrôleur, qui gérera l’affichage du menu à l’utilisateur ainsi que les entrées de celui-ci. Le fils quant à lui va tout d’abord s’initialiser en fermant tous les pipes qui lui sont inutiles avant de rentrer dans une boucle infini, et d’attendre une instruction.

Étape 3: Gestion de la communication entre processus

Tout processus fils s’initialise au départ en fermant tous les descripteurs de fichiers dont il n’a pas besoin , ainsi il ne peut lire que dans le pipe le reliant à node x-1 et il ne peut qu'écrire dans celui le reliant à node x+1. Cependant il y a une exception, le processus 0 lui lira dans le pipe le reliant à node N-1, et par inversement le processus N-1 écrira dans celui de node 0. Ainsi l’information se propagera dans l'anneau.

Exécution du programme:

Pour exécuter le programme, il faut se mettre dans le répertoire **src** puis compiler le fichier source c main.c avec la commande make qui produira l'exécutable et par la suite lancez al commande ./main <N> Où N est un entier positif supérieur ou égal à 2 représentant le nombre de processus à créer lors de l'exécution.

```
# ⏩ Important pour exécuter le programme !

Sur la ligne de commande depuis un terminal, lancez:


```sh
   cd src          #se positionner dans le répertoire src
   make            #compiler le programme
   ./main N        #exécuter le programme: N>=2
```


```