Jules NOEL
Gabriel NOURRIT

RAPPORT PROJET DICTIONNAIRE DE DONNEES

Programmation Système

SOMMAIRE

I / INTRODUCTION

Pour introduire notre projet nous allons parler des objectifs du projet ainsi que la finalité attendue pour celui-ci.

L'objectif principal de ce projet était de réaliser un dictionnaire permettant de stocker et de consulter des données. Toutes les données de notre dictionnaire seront de la forme chaîne de caractères. Notre dictionnaire sera représenté sous la forme d'une liste chaînée de couples clé valeur. La valeur sera la chaîne de caractère que nous voulons stocker et la clé sera associé à cette chaîne et permettra de retrouver notre donnée lors d'une recherche.

Dans ce dictionnaire nous pourrons utiliser trois commandes :

- Set : Commande qui permet de renter une nouvelle donnée dans notre dictionnaire. Pour cela il nous faut rentrer la valeur de notre donnée et sa clé associée.
- Lookup : Commande qui permet de rechercher une donnée dans notre dictionnaire. Pour cela il nous suffit de donner une clé et le programme nous renverra la donnée associée.
- Dump : Commande qui permet d'afficher l'intégralité de notre dictionnaire.

Pour gérer ce dictionnaire nous utiliserons des processus père-fils ainsi que des tubes permettant la communication entre ces processus.

La finalité attendue de ce projet est de pouvoir rendre un programme, codé en langage C, simulant le fonctionnement de ce dictionnaire. Le code de notre programme devra être structuré, facilement lisible. Toutes nos sources devront être organisés et notre programme sera fourni avec un Makefile.

II / Le programme du point de vue technique

1 / Choix de conception et de structuration

a) Structuration du programme en processus

Pour réaliser notre programme nous devions réaliser un système avec N processus nœud et un controler. Nos N processus seront reliés en anneau, c'est-à-dire que le processus Ni communique avec le processus Ni-1 en lecture et Ni+1 en écriture. Dans les cas particuliers du premier processus (1), ce processus pourra communiquer avec le processus N en lecture et 2 écriture. Enfin dans le cas du processus N, il pourra communiquer avec le processus N-1 en lecture et 1 en écriture. Chaque processus aura les fonctions set, lookup et dump décrites plus haut.

b) Communication entre les processus

Pour pouvoir communiquer les processus utiliseront des tubes. Il y aura en tout N tubes plus le tube général reliant tous les processus nœuds au controler. Le controler pourra écrire dans tous les tubes sauf le tube général, et pourra seulement lire dans le tube général.

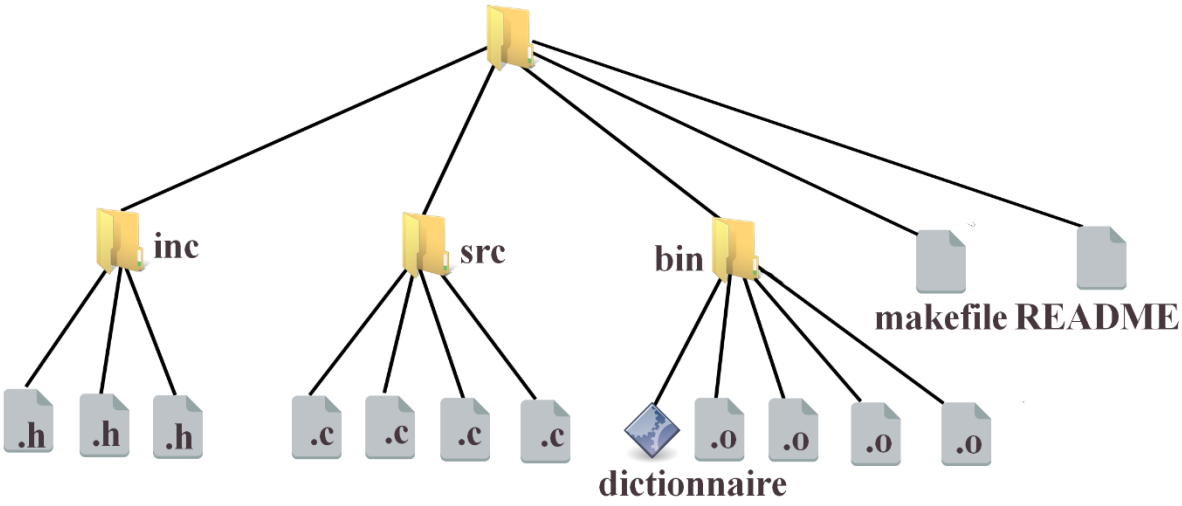
c) Structuration générale du programme

Afin d'organiser tout ce système nous avons décidé d'organiser notre programme en quatre entités :

- main.c : fichier contenant le programme principal à exécuter
- utilitaire.c : fichier contenant les fonctions dites ''utilitaires'' comme par exemple une fonction permettant de nettoyer une chaine récupérée en commande et que l'on va faire circuler dans les tubes. Ce fichier est accompagné du fichier utilitaire.h contenant les définitions des fonctions de utilitaire.c.
- table.c : fichier contenant toutes les fonctions sur le dictionnaire (set, lookup, dump), ce fichier est accompagné du fichier table.h contenant les définitions des fonctions de table.c.
- processus.c : fichier contenant toutes les fonctions concernant les processus, c'est-à-dire les fonctions permettant de créer les processus nœuds, la fonction simulant le comportement du controller ou encore les fonctions simulant les comportements des

processus nœuds. Ce fichier est accompagné du fichier processus.h contenant les définitions des fonctions de processus.c.

d) Structuration des fichiers sources



Afin de bien structurer notre projet nous avons décidé d'organiser en trois dossiers :

- src : contient tous les fichiers sources de notre code (main.c processus.c table.c utilitaire.c)
- inc : contiet tous les headers (processus.h table.h utilitaire.h)
- obj : contient tous les fichiers objets crée lors de la compilation (main.o processus.o table.o utilitaire.o)

Ensuite nous avons notre makefile dont nous parlerons plus tard, et notre exécutable qui sera créé lors de la compilation.

2) Fonctionnement du programme

Dans cette partie nous allons détailler le fonctionnement du programme. Pour commencer nous allons parler du format des messages transmis entre les nœuds et le controler. Nous avons trois types de messages :

- Message dans le cas d'un set :
  commande_utilisateur clé valeur?
  Commande_utilisateur : commande choisie par l'utilisateur (dans ce cas 1)
  Clé : clé choisie par l'utilisateur (int)
  Valeur : mot choisi par l'utilisateur(char*)
  ? : caractère pour définir la fin du message

- Message dans le cas d'un lookup :
  commande_utilisateur clé 0?
  commande_utilisateur : commande choisie par l'utilisateur (dans ce cas 2)
  Clé : clé choisie par l'utilisateur (int)
  0 : Caractère de remplissage
  ? : caractère pour définir la fin du message
- Message dans le cas d'un dump :
  commande_utilisateur 0 0?
  Commande_utilisateur : commande choisie par l'utilisateur (dans ce cas 3)
  0 : caractère de remplissage
  ? : caractère pour définir la fin du message

Lorsque l'on lance le programme, celui-ci appelle la fonction main. La première chose que l'on va faire est de créer les tubes grâce à une boucle et de les stocker dans un tableau afin de les réutiliser plus tard.

Ensuite nous allons définir le un comportement lors de la réception d'un signal SIGUSR1. Après nous allons créer tous les fils grâce à la fonction créer_fils(). Dans cette fonction nous allons créer tous les fils grâce à un fork dans une boucle, afin de ne pas créer de petits fils nous allons définir un comportement à chaque fils (fonction premierFils() et autreFils()) et les mettre en pause en attente d'un signal. Ensuite on ferme tous les pipes, on lance l'affichage et le controler du programme.

Le controler du programme va demander à l'utilisateur de choisir entre set, lookup, dump et exit. On récupère le choix de l'utilisateur grâce à la fonction scanf. Ensuite en fonction du choix de l'utilisateur nous allons exécuter des tâches différentes :

- Set : Dans ce cas-là nous demandons à l'utilisateur la clé et la valeur à stocker. Nous récupérons toujours ces valeurs grâce à scanf. Ensuite nous allons regrouper toutes ces informations dans une chaîne de caractère, nous verrons comment elle est formée plus tard. Enfin nous allons envoyer cette chaîne au premier fils grâce à son tube.
- Lookup : L'éxécution est la même que celle du set sauf que l'on demande seulement une clé à l'utilisateur.
- Dump : Dans ce cas-ci nous allons envoyer un message à tous les nœuds grâce à leur pipe, afin que chaque nœud affiche son dictionnaire.
- Exit : Nous allons tuer tous les nœuds en leur envoyant le signal SIGINT grâce à la fonction signal. Ensuite le controler va vérifier que tous les fils sont mort grâce à la fonction wait(). Enfin nous allons fermer les tubes et quitter le programme.

Ensuite le controler va lire le message envoyé par un des fils dans le tube général, et afficher ce message.

Le controler va tourner en boucle jusqu'à que l'utilisateur décide d'exit.

Maintenant nous allons décrire le comportement des nœuds lorsqu'ils reçoivent un signal.

Pour décrire le comportement des nœuds nous avons fait deux fonctions, une qui décrit le comportement du premier fils et une qui décrit le comportement des autres fils. Ces deux fonctions ont le même comportement, seul les tubes à ouvrir sont différents.

La fonction autreFils(numéro) possède une boucle while(1) qui permet de mettre le nœud en boucle jusqu'à qu'on lui dise de se terminer (grâce au signal SIGINT). Dans cette boucle on décompose le message reçu grâce à la fonction sscanf(), on récupère donc le cas dans lequel on est (1,2 ou 3), on récupère la clé et la valeur rentrées par l'utilisateur.

Ensuite on regarde le cas, si le cas est 1 ou 2 (set ou lookup), on vérifie si c'est au tour du nœud d'exécuter la commande, grâce au modulo. Si c'est au nœud de faire la commande, on regarde le cas, si le cas est lookup on regarde dans notre table si on a la valeur recherchée, si la valeur recherchée n'est pas dans la table on affiche que la valeur n'a pas été trouvé, sinon on affiche la valeur trouvée. Si on est dans le cas du set on va stocker la valeur dans notre table et écrire au controler et aux autres nœuds que l'on a bien stocké la valeur. Si ce n'est pas à nous de faire la commande on transmet le message au nœud suivant. Enfin si le cas est égal à 3 on affiche la table. Puis le nœud se met en pause jusqu'au prochain signal du controler.

3) Compilation et exécution

a) Compilation

Pour que la compilation soit simple nous avons créé un makefile. Celui-ci nous a pris un certain temps afin de le confectionner de manière la plus optimisée possible. Nous nous servons de toutes les notions vues en cours, ainsi que des pistes d'améliorations citées à l'oral. En effet :

- Il récupère les sources et les headers de manière générique grâce à wildcard.
- Il génère le nom des fichiers objets grâce aux noms préalablement chargé des sources
- Il déclare les options pour le compilateur de manière conditionnelle → si on ne précise pas d'options à make il ne prend que l'option Warnings All → sinon il définit selon ce qu'on lui donne : une variable DEBUG et/ou une variable NBNOEUD qui permettra au programme de savoir s'il crée un nombre de nœuds par défaut ou pas, et/ou s'il doit activer ou non le mode de débogage.
- Puis il utilise la règle implicite .c.o pour générer les fichiers objets et l'exécutable.
- Il place les fichier objets générés ainsi que l'exécutable dans le fichier bin.
- On y a mis une règle de dépendance des .o sur les .h afin que la modification des .h soit prise en compte quand on make sans supprimer les .o
- Puis il a une cible .PHONY qui permet de reconstruire les dépendances des cibles qu'elle contient, cela évite que les fichiers qui ont les noms de ces cibles dans le répertoire courant soit exécutés.

- Enfin il gère le lancement du programme, un nettoyage plutôt complet du répertoire ainsi que sa compilation.

Lors de la compilation on peut choisir le nombre de fils que contiendra notre controler ainsi que si on veut lancer le programme en mode debug. En effet, le mode debug une fois active permet de placer des breakpoint dans le programme en mettant une fonction nomme etat(); quand le programme passe dessus il va ouvrir un nouveau terminal dans lequel sont identifiés les processus en vie en liens avec notre programmes, puis 1 a 1 on voit les fils de ces programmes et les tubes qu'ils ont ouvert en pressant une touche du clavier dans le nouveau terminal.

b) Exécution

Pour lancer notre programme il suffit de lancer un petit script que nous avons écrit. Ce script demande le nombre de nœud souhaité ainsi que si on veut lancer le programme en mode débogage. Ensuite, il réalise la commande make avec les valeurs rentrées et lance le programme.
Lors de l'exécution de notre code nous pouvons voir que notre programme fonctionne comme voulu. En effet nous pouvons :

- Ajouter une valeur à notre dictionnaire
- Rechercher une valeur à notre dictionnaire
- Afficher notre dictionnaire
- Quitter le programme facilement et sans faire de zombies

III / Le programme du point de vue personnel

Ce projet, avait un objectif simple et plutôt clair. En soit, nous n'avons pas vraiment rencontré de difficultés à le comprendre. Cependant, nous en avons quand même rencontrés certaines. D'abord, sur le plan organisationnel, le partage et les échanges de fichiers au sein de notre binôme ne fût pas toujours pratiques. Il nous a fallût un temps pour mettre en place une solution efficace, qui permettent une répartition du travail, et une collaboration facilitée. Techniquement, le manque de recul nécessaire à une assimilation complète vis-à-vis de la notion de processus en général, a généré une source d'erreurs conséquentes. Par exemple la gestion de l'exécution parallèle des nœuds et du controler lors d'un dump nous a posé un peu de soucis.

La mise en place de solutions s'est faite de manière rigoureuse. Le fer de lance de notre processus fût la documentation. De tous types, du man Linux à des cours disponibles sur internet en passant par des forums. Il nous fallait comprendre de manière quasi-exhaustive afin d'être efficace et de ne pas reproduire les mêmes erreurs. Notre partage de connaissances et de réflexion ainsi que la recherche de différents points de vus de personnes d'autres binômes fût également une force. Notre volonté et notre capacité à nous relayer le projet, ont été un axe majeur de notre réussite.

IV / Conclusion

Ainsi nous pouvons voir que notre projet est assez abouti, en effet nous avons réussi à implémenter un programme qui répond aux objectifs. Nous avons respecté le cahier des charges :

- Dictionnaire sera représenté sous la forme d'une liste chaînée de couples clé valeur
- Nous pouvons faire 4 commandes (set, lookup, dump, exit)
- Utilisation de processus
- Structure en anneau
- Communication entre les processus par des tubes
- Pas de petits fils crée
- Seulement deux tubes ouverts par nœud

Ensuite d'un point de vue personnel nous pouvons dire que ce projet nous a permis de gagner en compétences dans le langage C, et surtout en programmation système. En effet, nous appréhendons mieux la notion de processus, signaux et tubes. Même si ce projet n'a pas toujours été facile, nos recherches sur internet ainsi que les discussions avec les autres binômes nous a permis de réaliser le projet dans le temps, tout en prenant soin du code et de sa structure.