Rapport
Projet PIM, stockage et exploitation de tables de routage

Introduction

Le but de ce projet est de construire un routeur. À partir d'un fichier table contenant les destinations, leur masque et leur interface, et d'un fichier paquet contenant les destinations à router, nous devons obtenir les interfaces. Les destinations et interfaces seront enregistrées dans un fichier résultat

Pour optimiser le routage, la création d'un cache est envisagée. Le cache sera codé sous la forme d'une liste chainée ou d'un arbre. De plus, différentes politiques de mises à jour du cache seront envisagées, FIFO, LRU et LFU.

Table des matières

Convention d'écriture

Le type de données sera en **gras** et précédé d'un **T_** .
Les fonctions *fct_* et procédures *prc_* seront en italique.

Architecture

1. Fonctions et procédures outils

a. Définition des types de données

Les données fournies dans les fichiers .txt seront d'abord enregistrées comme des **Unbounded_String**.

Dans les fichiers sources, les destinations, qui sont des adresses IP, sont composées de 4 entiers séparés par des points, ils seront donc représentés en Ada par un tableau de 4 entiers, **T_adresse_IP.** Les masques seront également des **T_adresse_IP** mais composé seulement de 0 et de 1. Exemple *:* destination1 255.40.32.12 -> [255, 40, 32, 12]

Les éléments de la table **T_table_elt** sont composés d'une destination **T_adresse_IP**, d'un masque **T_adresse_IP** et d'une interface **Unbounded_String**. Il s'agit donc d'un enregistrement contenant ces trois éléments.

Type de cache

- LCA : on importe le package LCA en instancient le type générique **T_element** avec **T_table_element**
- Arbre

Un type **T_bin_8** est également créé, ce tableau de taille 8 est composé de 0 et de 1, il représente l'écriture binaire d'un entier du masque. **T_adresse_bin** est le tableau de taille 4 de **T_bin_8** représentant le masque en binaire.

b. Module relatif à la gestion des adresses IP

Nous allons maintenant voir l'ensemble des fonctions et procédures qui permettent la gestion des adresses IP.

- *prc_afficher_adresse* : procédure affichant une adresse IP **T_adresse_IP**
- *prc_afficher_adresse_bin :* procédure affichant une adresse IP écrite en binaire **T_adresse_bin**
- *fct_convertir_10_2* : cette fonction convertit un entier écrit en base 10 **Integer** en entier écrit en base 2 **T_bin_8**
- *fct_convertir_10_2_adresse* : fonction convertissant une adresse écrite en base 10 **T_adresse_IP** en base 2 **T_adresse_bin**
- *fct_convertir_2_10* et *fct_convertir_2_10_adresse* : les fonctions inverses des deux précédentes
- *fct_et_bin :* fonction qui effectue un & bit à bit entre deux entiers représentés en binaire **T_bin_8**, elle renvoie l'entier **T_bin_8** obtenu.
- *fct_masque_bin :* fonction qui applique un masque **T_adresse_bin** à une destination **T_adresse_bin,** elle renvoie la nouvelle destination
- *fct_egalite :* cette fonction compare deux adresses IP **T_adresse_bin** et renvoie un **boolean**
- *fct_comp :* cette fonction prend en arguments un élément de la table **T_table_elt** et une destination **T_adresse_IP**, elle renvoie un booléen **Boolean** si l'adresse dans la table convient.
- *fct_taille_masque :* fonction qui renvoie la taille **Integer** du masque **T_adresse_bin**.
- *fct_masque_allonge :* cette fonction renvoie un masque **T_adresse_IP** allongé de 1 bit par rapport à celui entré en argument **T_adresse_IP.**
- *prc_new_masque et fct_new_masque :* procédure et fonction qui permettent d'obtenir le nouveau masque à ajouter au cache. Pour obtenir le bon masque, on parcourt tout le fichier table et on regarde s'il n'y a pas de conflit ie si on ajoute ce masque au cache

on obtiendrait le même résultat que si nous n'avions pas de cache, avec le cache la réponse est simplement plus rapide.

c. Gestion du cache LCA

En fonction de la politique du cache, la procédure pour ajouter des éléments dans le cache est différente. Il s'agit en réalité de savoir quel élément supprimer lorsque le cache est plein.

i. Politique FIFO

- *prc_ajouter_FIFO :* on prend en entrée le cache **T_LCA** et l'élément à ajouter **T_table_elt.** Si le cache est plein alors on supprime le premier élément de la liste. Ainsi il reste toujours une place de libre pour ajouter le nouvel élément dans le cache. On ajoute ainsi l'élément en queue.

ii. Politique LRU

- *prc_ajouter_LRU :* on prend en entrée le cache **T_LCA**, l'élément à ajouter **T_table_elt** et l'indice **integer** de la destination qui a permis de router le paquet s'il appartient au cache**.** Si le cache est plein et que nous avons bien utilisé un élément dans le cache pour router le paquet alors on déplace cet élément en queue puis on supprime la tête. Et enfin on enregistre le nouvel élément dans le cache en queue.

iii. Politique LFU

- *Prc_ajouter_LFU :* les variables sont le cache **T_LCA** et l'élément qui vient d'être routé **T_table_elt**. Si le cache est plein alors on supprime selon la politique LFU (package LCA_table). On ajoute le nouvel élément avec le compteur mis à jour. Ce compteur nous permet de connaître la fréquence d'utilisation de la destination afin de supprimer le bon élément.

d. Gestion du cache sous forme d'arbre

Cette fois le cache est sous forme d'un arbre. Nous avons repris la même structure que pour routeur_LL, en modifiant simplement le type de cache, nous avons utilisé le module Cache_LA.

e. Procédure d'affichage et autres fonctions

- *prc_afficher_cache* : affiche tous les éléments du cache
- *prc_afficher_txt* : affiche tous les éléments de la table, d'un fichier .txt en général
- *fct_arrondi :* elle arrondi un chiffre représenté par un flottant **float** en un entier sous forme de chaine de caractère **Unbounded_String**.

f. Déclaration des données et paramètres par défaut

Les paramètres par défaut sont les suivants :

- La taille du cache : 10
- La politique : FIFO
- Le fichier table : table.txt
- Le fichier des paquets à router : paquet.txt
- Le fichier dans lequel les résultats seront inscrits : resultats.txt
- Afficher les statistiques du cache à la fin du traitement des paquets : oui

2. Programme

a. Gestion de la ligne de commande

Dans un premier temps, on récupère toutes les informations dans la ligne de commande. Pour cela on effectue une boucle pour sur le nombre d'argument entré en ligne de commande. Si

l'utilisateur commet une erreur alors un message d'erreur personnalisé sera affiché en fonction et le paramètre par défaut sera choisi.

b. Enregistrement du fichier entrée table dans une LCA

Nous commençons par initialiser une liste vide. Puis nous parcourons le fichier table, grâce à une boucle répéter tant que tout le fichier n'a pas été lu. On ajoute au fur est à mesure les elements **T_table_elt** contenant la destination, le masque et l'interface.

c. Traitement du fichier paquet

Afin de traiter le fichier paquet, nous avons utilisé une boucle répéter tant qu'il reste des destinations à traiter (que le fichier paquet n'a pas été entièrement parcouru) ou que le mot fin n'est pas apparu. Il s'agit maintenant de traiter les lignes du paquet : soit il s'agit d'une destination, soit d'une commande.

i. Traitement d'une destination

Dans un premier temps, on récupère l'adresse de la destination que l'on enregistre dans un nouveau **T_table_element.destination.** Il n'y a pas encore de masque donc la taille du masque est initialisé à zéro. On parcourt ensuite avec une boucle pour l'ensemble du cache pour voir si une destination convient.

Pour savoir si une destination convient, on applique le masque à la destination du cache et à la nouvelle destination, s'ils sont égaux et que la taille du nouveau masque est supérieure à la précédente alors on enregistre l'interface et le masque.

Si nous n'avons pas trouvé de destination qui convienne dans le masque, taille du masque est toujours égale à zéro, alors on parcourt la table.

Enfin on ajoute la nouvelle destination avec le masque et l'interface au cache si la taille du masque finale est différente de zéro (ie l'interface n'est pas celle par défaut). Pour ajouter le nouvel élément au cache, on utiliser les procédures ajoutées en fonction de la politique choisie.

On affiche ensuite la destination avec son interface dans le terminal. On ajoute la destination et l'interface dans le fichier resultat, sur une nouvelle ligne.

ii. Traitement d'une commande

Il suffit d'appliquer la commande lue dans le fichier paquet.

- Table : on fait appel à *prc_afficher_txt* pour afficher la table
- Stat : idem que j. Affichage des statistiques du cache (ci-dessous)
- Cache : affiche le contenu du cache ainsi que sa politique
- Fin : on arrête de traiter les paquets, fin du répéter.

d. Affichage des statistiques du cache

Après avoir traité tout le fichier paquet ou à la suite de la lecture du mot fin, on affiche les statistique du cache si tel a été demandé. En commençant par le défaut de cache, nombre de fois où le cache n'a pas pu fourni d'interface correspondant à une destination ; le nombre de demande de destination faite au cache ; et le rapport défaut de cache sur nombre de demande.

e. Récupération de l'heure

Afin de comparer la différence d'efficacité entre nos différents types de cache, nous avons décidé d'ajouter une variable qui calcule le temps d'exécution mis par le programme pour traiter le paquet.

Module LCA

Le module LCA est utilisé dans les deux programmes. Dans le routeur_LL, les listes chainées sont utilisées pour enregistrer la table, les paquets à router ainsi que le cache. Dans le

routeur_LA, seuls les fichiers table et paquet seront représentés par des listes chainées, le cache sera sous forme d'arbre (voir module arbre).

Le package lca_table est un module générique, il suffira d'instancier **T_element** pour pouvoir l'utiliser. Voici les foncions qu'il contient :

- *Initialiser :* cette procédure permet de créer une nouvelle liste
- *Est_Vide :* fonction booléenne qui renvoie si la liste est vide ou non
- *Enregistrer :* procédure qui ajoute un élément **T_element** avec sa fréquence **integer** dans la Sda **T_LCA.** L'ajout se fait en queue. Cette procédure est donc récursive.
- *Fct_taille :* fonction qui renvoie la taille de la Sda **T_LCA** de manière récursive.
- *Prc_deplacer_i_fin* : procédure qui déplace le i ème élément de la Sda à la fin.
- *Prc_supprimer :* procédure qui supprime le premier élément de la liste
- *Prc_supprimer_dernier* : procédure qui supprime le dernier élément de la liste
- *Prc_supprimer_LFU :* procédure qui supprime un élément de la liste selon la politique LFU
- *Vider* : procédure qui vide la liste
- *Pour_Chaque :* procédure qui applique un traitement à chaque couple d'une liste chainée

Module Cache_LA

Le module Cache_LA est uniquement utilisé dans le routeur LA, il permet de traiter le cas où le cache est un arbre **T_Cache_LA**. Voici les fonctions qui seront utilisées dans routeur_LA :

- *Prc_initialiser* : initialise un cache de type **T_cache_LA**
- *Fct_EstVide* : renvoie le booléen vrai si la liste est vide, faux sinon
- *Prc_Ajouter* : procédure qui prend en argument un cache et une route à ajouter à ce cache. Cette procédure est clé car elle nous permet d'ajouter la route au cache.
- *Fct_taille* : fonction qui renvoie la taille du cache **T_Cache_LA**
- *Fct_ASupprimer :* fonction qui renvoie la route à supprimer du cache
- *Prc_Supprimer :* procédure qui supprime l'élément ancienne_route **T_Routes** du cache LA
- *prc_rechercher :* cette procédure permet d'obtenir l'interface de destination d'un paquet si une route correspond dans le cache.

Modules de test

À chaque fois que nous codions des nouvelles fonctions ou procédures, nous les testions indépendamment avant de les rajouter dans le code principal. Les modules de test généraux ont plusieurs objectifs :

- Vérifier que les interfaces renvoyées sont les bonnes
- S'assurer que les éléments supprimés dans le cache correspondent bien à la politique choisie

Nous avons ainsi travaillé avec différents tables et paquets pour tester tous les cas particuliers, notamment la cohérence du cache. Des tests ont également été effectués sur la ligne de commande pour vérifier que l'utilisateur ne pouvait pas demander des choses incohérentes au programme. Cela nous a également permis de contrôler les différents messages d'erreur.

Enfin, une fois les différents types de cache et politique codés, nous avons eu le loisir de comparer les résultats ainsi que les temps d'exécution.

Difficultés rencontrées et solutions apportées

- Application des masques aux différentes adresses IP

Pour résoudre cette difficulté, nous avons écrit des fonctions qui transforment un entier d'écriture décimal en écriture binaire. Puis on compare bit à bit, on obtient ainsi la nouvelle adresse IP.

Avoir codé ces différentes fonctions, nous a permis de mieux visualiser les actions de comparaison et d'application de cache.

- Obtention du meilleur masque à ajouter au cache

Nous parcourons toute la table, et si une ambiguïté est rencontrée alors on allonge le masque en prenant celui de la table ainsi il ne peut y avoir de problèmes.

- Extraction des différentes informations contenues dans les fichiers .txt pour qu'ils correspondent aux types manipulés dans nos programmes

Des boucles pour, la gestion des . et des ' ' (espace) a permis de récupérer les informations contenues dans les fichiers .txt.

- Gestion du terme fin dans les paquets

Afin de pouvoir arrêter le traitement des paquets dès que nous le souhaitions, nous avons utilisé une boucler répéter avec plusieurs conditions d'arrêt : soit le fichier paquet est vide, soit le mot fin a été rencontré.

- Traitement des exceptions lorsque l'utilisateur ne rentre pas les bonnes informations dans la ligne de commande

Nous avons rajouté des si dans le selon afin de traiter les différentes erreurs que l'utilisateur serait amené à faire.

- Le module arbre de manière générale

La gestion du module arbre a été plutôt complexe et a nécessité de créer des modules auxiliaires pour gérer les différents problèmes que nous avons pu rencontré.

Organisation

1. Raffinage

Robin a écrit la partie sur la ligne de commande. Laura s'est occupée du cache par liste chainée et des différentes politiques. Henri a énormément travaillé sur les raffinages: cache par arbre, homogénéisation des différentes parties… Relecture finale par Laura.

2. Module LCA

Une première version du module a été écrite pour le rendu de décembre afin de pouvoir enregistrer les éléments contenus dans le fichier table. Cette première version écrite par Laura a été complétée par Robin en janvier avec l'ajout du champ fréquence dans l'enregistrement **LCA.** Cet ajout était nécessaire pour traiter le cas du cache avec une politique LFU, pour les autres politiques ce champ gardera la valeur 0.

3. Premier rendu

Les modules relatifs à la gestion des adresses IP ont été en grande partie codés par Laura ainsi que l'application du masque et l'obtention de l'interface.

L'extraction des informations contenues dans les fichiers .txt et l'écriture dans ces fichiers par Robin. Le selon a également été codé par Robin

4. Second rendu

Pour le cache LL, la politique LFU a été codée par Robin, LRU et FIFO par Laura.

Le cache LA avec le module LA ont quant à eux été codés par Henri et complété par Laura.

Bilan technique

À la fin de ce projet, nous avons un routeur qui fonctionne avec un cache sous forme de liste chainée et d'arbre. Le raffinage écrit au départ est certes différent de notre code final (50%), il nous a été cependant très utile pour nous répartir le travail ainsi que pour savoir les modules qui seraient réutilisés dans les différentes parties. Au terme de ces quelques semaines de travail sur ce projet, nous pensons avoir réussi à remplir le contrat, l'utilisateur peut faire ses demandes via la ligne de commande et obtient les résultats du routage dans un fichier .txt. S'il commet des erreurs alors le programme est capable de le lui dire.

Nous pourrions améliorer le programme en ajoutant le plus petit masque possible au cache mais le temps de calcul serait beaucoup plus important car il faudrait parcourir de nombreuses fois la table. Il s'agit ici d'une piste d'amélioration : obtenir le meilleur cache possible sans ajouter un trop long temps de calcul.

De plus de nombreuses fonctions et procédures « outils » sont codées dans le programme principal routeur_LL, il pourrait être intéressant de les écrire dans des package séparés puis de les appeler dans le programme principal comme c'est en grande partie le cas pour le routeur_LA.

Bilans personnels

Robin

Le projet m'a permis de mieux comprendre comment les différentes parties du cours qui au premier abord me paraissaient distantes peuvent être mises en commun. J'ai bien aimé m'occuper de la partie ligne de commande car c'était tout nouveau et je trouve ça intéressant de s'occuper de l'interaction homme/programme.

Je n'ai pas eu de difficultés à travailler en groupe et la répartition du travail s'est faite très naturellement. Pour la conception, l'essentiel a été réalisé durant les séances dédiées en cours et pour la mise au point, j'ai pris une quinzaine d'heure pour réaliser l'ensemble du corps du programme et une dizaine pour comprendre et mettre au point la ligne de commande.

Henri

Ce projet fut, à mon sens, une bonne application des notions vues en cours permettant d'aller plus loin dans leur compréhension (plus loin que ne l'ont permis les TPs). J'ai personnellement pris plaisir aussi bien dans la conception des solutions que dans leur mise en œuvre/implémentation. Si ce projet m'a permis d'approfondir mes connaissances en ADA et en programmation, il a aussi été une sorte de défis - j'ai plusieurs été confronté à des problèmes majeurs -.

Je pense avoir passé au total environ 15h sur la conception des solutions et tout autant sur leur implémentation.

Laura

Ce projet de création d'un routeur était très intéressant. L'une des tâches les plus dures a été de bien comprendre le sujet à son commencement. L'ensemble du projet m'a permis de mettre en application les différentes notions que nous avions étudiées en cours. Il est également très complet et nous fait utiliser différentes spécificités du langage ADA.

De plus, travailler en groupe sur un programme informatique nécessite une bonne communication (choix des différents types de données, structure du programme, mise en commun…). J'estime que nous avons réussi à bien nous répartir le travail et à communiquer sur le projet afin de le mener à son terme.

En plus des 10 heures de projet en cours, je pense avoir travaillé sur le routeur 30 heures en plus, en considérant la conception, l'implantation, la mises au point du programme et le rapport.