Rapport de projet

Routage

Mathis Sigier, Benoit Maggion, Aléxy Fièvet

Programmation Impérative

ENSEEIHT

Janvier 2023

Plan du rapport

1. Introduction du sujet
2. Raffinages
3. Architecture générale du code final
4. Détails sur l’architecure du code
5. Difficultés de mise en place
6. Conclusion sur le projet

I - Introduction

Le

routage en informatique est un processus d’acheminement de données à travers un réseau vers une interface liée aux informations données. C’est la table de routage qui structure l’ensemble des informations pour l’acheminement. C’est une façon efficace et rapide d’envoyer un grand nombre d’informations vers des interfaces choisies.

Dans le cadre de notre enseignement de la matière programmation impérative. Nous avons eu à réaliser un code complet et efficace de l’utilisation d’un routeur, d’une table de routage et éventuellement du cache associé.

Nous avons utilisé le language Ada pour écrire notre code.

Nous avons un mois et demi pour réaliser ce projet en trio.

Après avoir pris connaissance du sujet, nous avons rapidement commencé à écrire le raffinage du projet pour pouvoir avoir une vision d’ensemble du code que nous allions créer avec Ada. Les raffinages constituent donc une base que nous avons suivi tout au long de la rédaction des différents codes.

Les objectifs généraux du projet étaient de pouvoir extraire la table de routage inscrite dans un fichier texte pour l’implémenter dans une LCA, être capable de traiter les différentes destinations des paquets qui étaient aussi inscrites dans un fichier texte et générer un fichier résultat.

Dans un premier temps nous avons codé un routeur simple ne fonctionnant sans cache. Puis nous avons coder deux routeurs utilisant ds caches : un routeur disposant d’un cache sous forme de liste chainée, l’autre utilisant un arbre.

II - Raffinages

Routeur Simple :

**R0 :**  Afficher la bonne destination pour un paquet reçu.

**R1** : **Comment** :  “Afficher la bonne destination pour un paquet reçu.” ?

* Mettre la table de routage sous forme de liste chainée.
* Choisir la destination la plus adaptée.
* Afficher cette destination.                                                       destination\_finale : out.

**R2** : **Comment** : “Mettre la table de routage sous forme de liste chainée.”?

* Classer les valeurs de la table par type.                  destination; masque; interface : in
* Convertir l’adresse de la destination en binaire.                     destination\_binaire : out.
* Convertir le masque en binaire.                                              masque\_binaire : out.
* T\_Table de type Pointeur sur liste\_chainée T\_route.

**R3 : Comment** : “Table de type Pointeur sur liste\_chainée route.” ?

Type T\_route est :

destination\_binaire : mod 2\*\*32

masque\_binaire : mod 2\*\*32

interface : string

suivant : T\_route

**R2** : **Comment** : “Choisir la destination la plus adaptée.” ?

* Recupérer l’adresse IP du paquet.                                          adresse : in.
* Convertir l’adresse IP du paquet en binaire.                           adresse\_binaire : out.
* Sélectionner la route la plus adaptée.

**R3** : **Comment** : “Sélectionner la route la plus adaptée.” ?

* Classer les routes en fonction de la taille des masques (du plus long au plus court.)                                                                                                  taille\_masque\_i : out.
* Sélectionner la première route correspondant (avec le bon nombre de bits à comparer (représenté par la taille du masque)).                      destination\_finale : in.

**R4** : **Comment** : “Classer les routes en fonction de la taille des masques.” ?

**Pour** i allant de 1 à nombre\_masque **faire:**

reste = 0

masque2 = masque

taille\_masque\_i = 32

**Tant que** reste = 0 et taille\_masque\_i >0 **Faire**

reste = masque2%2;

  masque2 = masque2/2

taille\_masque\_i = taille\_masque\_i - 1

Fin Tant Que

Fin Pour

**R4** : **Comment** : “Sélectionner la première route correspondant.” ?

**Pour** i allant de 1 à nombre\_route **faire**

Comparer l’adresse IP avec la destination.

**R5** : **Comment** : “Comparer l’adresse IP avec la destination.” ?

masque2 = masque

destination2 = destination

**Pour** j allant de 1 à taille\_masque **faire**

masque2 = masque2/2

destination2 = destination2/2

**Si** masque2 = destination2 **faire**

interface\_finale = interface

renvoyer interface\_finale

Sinon faire

renvoyer destination\_par\_defaut

Fin Si

Détails du cache :

Routeur LL avec les trois types de politiques:

**R0:** Utiliser un routeur LL qui propose les trois politiques FIFO, LRU et LFU

**R1: Comment “**Utiliser un routeur LL qui propose les trois politiques FIFO, LRU et LFU”?

Initialiser un routeur LL qui propose les trois politiques FIFO, LRU et LFU (Taille\_cache : Integer, Politique\_choisie : String, Statistique : Boolean, Table : Text\_file, Paquets : Text\_file, Résultats : Text\_file, TableR : LC, SD\_Cache : LC): **out**

Traiter les paquets Paquets : **in out**

Restituer les résultats Resultats : **in out**

**R2: Comment “**Initialiser un routeur LL qui propose les trois politiques FIFO, LRU et LFU”?

Récupérer les caractérisitiques, la politique et les données (Taille\_cache : Integer, Politique\_choisie : String, Statistique : Boolean, Table : Text\_file, Paquets : Text\_file, Résultats : Text\_file): **out**

Traiter l’ensemble des commandes reçues (Taille\_cache : Integer, Politique\_choisie : String, Statistique : Boolean, Table : Text\_file, Paquets : Text\_file, Résultats : Text\_file): **in**

Initialiser la table de routage sous forme de liste chaînée (TableR) : **out**

Initialiser la structure de données qui représente le cache (SD\_cache) : **out**

**R2 : Comment “**Traiter les paquets”?

**Pour** i **dans** 1..Nombre\_paquets **Faire** - - On suppose que l’on connait le nombre total de paquets dans le fichier Paquets

Traiter le paquet\_i Paquet\_i : **in**

FinPour

**R2: : Comment “**Restituer les résultats”?

**Pour** i **dans** 1..Nombre\_paquets **Faire**

**Ecrire**(Str(Paquet\_i)) - - On affiche le paquet demandé et l’interface de destination sur la même ligne

**Ecrire**(Resultats(i))         - - Affiche l’interface de sortie

Retour à la ligne

FinPour

**Si** Statistique = True **Faire**

Afficher les statistiques

Fin Si

**R3 : Comment “**Traiter le paquet\_i”?

Choisir la destination la plus adaptée dans le cache (Route\_cache : Boolean, Destination\_cache : String ): **out,** (Paquet\_i) : **in** - - Destination cache est l’interface d’arrivée

**Si** Route\_cache = False **faire** - - Pas de destination trouvée dans le cache

Choisir la destination la plus adaptée dans la table de routage (Paquet\_i :) **in**

Fin Si

Mettre à jour le cache en fonction de la politique choisie (Paquet\_i): **in, (**SD\_cache) : **in out**

**R3:** **Comment “**Récupérer les caractéristiques, la politique et les données” ?

Taille\_cache <- 10 - - On met dans un premier temps toutes les valeurs par défauts

Politique\_choisie <- “FIFO”

Statistique <- ‘-s’

Table <- table.txt

Paquets <- paquets.txt

Resultats <- resultats.txt

**R3: Comment “**Traiter l’ensemble des commandes reçues”?

Créer un tableau recensant l’ensemble des commandes (Tableau\_commandes) : **out,**

Numero\_argument <- 1

**Tant que** Numero\_argument < Nombre\_argument **Faire** - - Il y a maximum 5 arguments lors de l’appel de la fonction, on suppose Nombre\_argument connu

Traiter l’argument reçu Numero\_argument : **in**

Numero\_argument <- Numero\_argument +1

FinTantque

**R3: Comment** initialiser la table de routage sous forme de liste chaînée?

**Initialiser** (TableR) - - On initialise une liste chaînée vide (programme équivalent lors du mini projet 2)

**Pour** i dans 1..Nb\_lignes\_Table **Faire** - - On suppose que l’on connaît le nombre de lignes total du fichiers texte table

Enregistrer\_Ligne\_i

FinPour

**R3: Comment “**Initialiser la structure de données qui repéresente le cache”?

**Initialiser**(StructD\_Cache)

**R4 : Comment “**Mettre à jour le cache en fonction de la politique choisie”?

**Si** Taille(SD\_Cache)>=Taille\_cache **Faire**

**Selon** Politique\_choisie **Dans**

“FIFO” => Supprimer Donnee\_la\_plus\_ancienne(SD\_cache)

“LRU” => Supprimer Donnee\_moins\_recemment\_utilisee(SD\_cache)

“TFU” => Supprimer Donnee\_moins\_utilisee(SD\_cache)

FinSelon

FinSi

Implanter dans le cache le dernier paquet utilisé

**R4: Comment** “Afficher les statistiques”?

**R4 : Comment “**Choisir la destination la plus adaptée dans le cache”?

**R4 : Comment “**Choisir la destination la plus adaptée dans la table de routage”?

**R4: Comment “**Traiter l’argument reçu”?

**Selon** Tableau\_commandes(Numero\_argument) **Dans**

“-c” => (Taille\_cache <- Integer’Value(Tableau\_commandes(Numero\_argument+1))

“-p” => (Politique <- Tableau\_commandes(Numero\_argument+1))

“-s” => (Statistique <- True)

“-S” => (Statistique <- False)

“-t” => (Table <- Tableau\_commandes(Numero\_argument+1))

“-p” => (Paquets <- Tableau\_commandes(Numero\_argument+1))

“-r” => (Resultats <- Tableau\_commandes(Numero\_argument+1))

**R5 : Comment “**Implanter dans le cache le dernier paquet utilisé”

R5 : Comment

Routeur LA avec la politique LRU:

**R0:** Utiliser un routeur LA en politique LRU

**R1: Comment** Utiliser un routeur LA en politique LRU?

Initialiser un routeur LA en politique LRU (Taille\_cache : Integer, afficher\_statistiques: Boolean, Table : Text\_file, Paquets : Text\_file, Résultats : Text\_file, TableR : LA, SD\_Cache : LA): **out**

Traiter les paquets Paquets : **in out**

Restituer les résultats Resultats : **in out**

**R2: Comment “**Initialiser un routeur LA en politique LRU?

Récupérer les caractéristiques, et les données (Taille\_cache : Integer, Statistique : Boolean, Table : Text\_file, Paquets : Text\_file, Résultats : Text\_file): **out**

Traiter l’ensemble des commandes reçues (Taille\_cache : Integer, Statistique : Boolean, Table : Text\_file, Paquets : Text\_file, Résultats : Text\_file): **in**

Initialiser la table de routage sous forme de liste chaînée (TableR) : **out**

Initialiser la structure de données qui représente le cache (SD\_cache) : **out**

**R2 : Comment “**Traiter les paquets”?

**Pour** i **dans** 1..Nombre\_paquets **Faire** - - On suppose que l’on connait le nombre total de paquets dans le fichier Paquets

Traiter le paquet\_i Paquet\_i : **in**

Mettre à jour le cache en fonction de la politique choisie

FinPour

**R2: : Comment “**Restituer les résultats”?

**Pour** i **dans** 1..Nombre\_paquets **Faire**

**Ecrire**(Str(Paquet\_i)) - - On affiche le paquet demandé et l’interface de destination sur la même ligne

**Ecrire**(Resultats(i))         - - Affiche l’interface de sortie

Retour à la ligne

**Si** Statistique = True **Faire**

Afficher les statistiques

Fin Si

**R3 : Comment “**Traiter le paquet\_i”?

Choisir la destination la plus adaptée dans le cache (Route : Boolean, Destination\_cache : String ): **out,** (Paquet\_i) : **in** - - Destination cache est l’interface d’arrivée

**Si** Route = False **faire**

Choisir la destination la plus adaptée dans la table de routage (Paquet\_i :) **in**

Fin Si

**R3:** **Comment “**Récupérer les caractéristiques et les données” ?

Taille\_cache <- 10 - - On met dans un premier temps toutes les valeurs par défauts

Statistique <- ‘-s’

Table <- table.txt

Paquets <- paquets.txt

Resultats <- resultats.txt

**R3: Comment “**Traiter l’ensemble des commandes reçues”?

Créer un tableau recensant l’ensemble des commandes (Tableau\_commandes) : **out,**

Numero\_argument <- 1

**Tant que** Numero\_argument < Nombre\_argument **Faire** - - Il y a maximum 5 arguments lors de l’appel de la fonction, on suppose Nombre\_argument connu

Traiter l’argument reçu Numero\_argument : **in**

Numero\_argument <- Numero\_argument +1

FinTantque

**R3: Comment** initialiser la table de routage sous forme de liste chaînée?

**Initialiser** (TableR) - - On initialise une liste chaînée vide (programme équivalent lors du mini projet 2)

**Pour** i dans 1..Nb\_lignes\_Table **Faire** - - On suppose que l’on connaît le nombre de lignes total du fichiers texte table

Enregistrer\_Ligne\_i

FinPour

**R3: Comment “**Initialiser la structure de données qui repéresente le cache”?

**Initialiser**(StructD\_Cache)

**R4: Comment** “Afficher les statistiques”

Afficher le nombre de défauts de cache

Afficher le nombre de demandes de route

Afficher le taux de défaut de cache

**R4 : Comment “**Choisir la destination la plus adaptée dans la table de routage”?

Liste chainé

**R4: Comment “**Traiter l’argument reçu”?

**Selon** Tableau\_commandes(Numero\_argument) **Dans**

“-c” => (Taille\_cache <- Integer’Value(Tableau\_commandes(Numero\_argument+1))

“-s” => (Statistique <- True)

“-S” => (Statistique <- False)

“-t” => (Table <- Tableau\_commandes(Numero\_argument+1))

“-p” => (Paquets <- Tableau\_commandes(Numero\_argument+1))

“-r” => (Resultats <- Tableau\_commandes(Numero\_argument+1))

**R4 : Comment “**Implanter dans le cache le dernier paquet utilisé”

**R4 : Comment “**Choisir la destination la plus adaptée dans le cache”?

arbre

if cache\_est\_vide (in out booléen) then

* Initialiser le cache
* Trouver la destination
* Afficher la destination
* Gérer le cache selon une politique LRU

else

* Trouver la destination
* Afficher la destination
* Gérer le cache selon une politique LRU

**R5  : Comment “**Initialiser le cache”?

Initialiser le cache (racine)

**R5  : Comment “**Trouver la destination”?

se placer à la racine

while arbre(droite) != Null and arbre(gauche) != Null loop

if destination>arbre(centre) and arbre(droite) == Null then

comparer l’adresse IP avec la destination

elsif destination<arbre(centre) and arbre(gauche) == Null then

comparer l’adresse IP avec la destination

elsif destination>arbre(centre) then

aller\_a\_droite(arbre)

elsif destination<arbre(centre) then

aller\_a\_gauche(arbre)

else (if destination == arbre(centre))

comparer l’adresse IP avec la destination

end if

end loop

**R5  : Comment “**Afficher la destination”?

Afficher la destination

**R5  : Comment “**Gérer le cache selon une politique LRU”?

if taille\_cache=taille\_cache\_init then

supprimer la donnée la moins récemment utilisé

ajouter 1 à la ‘récentitude’ de toutes les données

end if

if destination == arbre(centre) then

réinitialiser sa ‘récentitude’ à 0

else

ajouter la donnée à la detination avec un ‘récentitude’ de 0

end if

**R6  : Comment “**supprimer la donnée la moins récemment utilisée"?

ind\_max = 0

for i in (1 .. Taille\_cache) do

if recentitude(arbre(i))>recentitude(arbre(ind\_max)) then

ind\_max = i

end if

end loop

supprimer(arbre(ind\_max))

Commentaires sur les raffinages

Ces raffinages constituent la base du code général qui a suivi dans notre projet.

Architecture générale du code final

Le

code général du projet est rédigé avec le language de programmation Ada. Dans un premier temps nous avons codé un routeur simple. Les différents modules créés ont servi au codage du routeur avec cache. La plupart des modules sont réutilisés dans le routeur avec cache.

Le routeur simple

Module table:

Dans ce module, nous définissons la liste chainée qui définira la table de routage. Il est composé des fonctions et procédures : Initialiser, Taille, Enregistrer, Supprimer, Vider, Route\_Presente, LE\_Masque, L\_Interface, Chercher\_Route.

*Initialiser* : est une procédure qui initialise une table vide.

*Taille* : est une fonction qui renvoie la taille de la table de routage.

*Supprimer* : est une procure prenant en argument la table que l’on souhaite modifier et la route que l’on souhaite supprimer de la table de routage.

*Vider* est une procédure qui supprime l’ensemble des routes de la table.

*Route\_Presente* est une fonction qui renvoie un booléen indiquant si une route est pésente dans la table ou non.

*Le\_Masque* est une fonction qui prend en paramètre une destination et retourne le masque associé à cette destination.

*L\_Interface* est une fonction qui prend en paramètre une destination et qui retourne l’interface de sortie associée à cette destination.

*Chercher\_Route* est une fonction qui renvoie l’interface de la route qui correspond à l’adresse ip du paquet envoyée selon les données des routes de la table de routage.

Module adresse ip:

Dans ce module, nous définissons le type adresse ip et l’ensemble des fonctions et procédures associée. Il y a aussi les fonctions et procédures liée à l’enregistrement et l’écrites des routes depuis ou vers un fichier texte.

*Le type adresse ip* est codé sur 32 bits. On regroupe ces 32 bits en 4 octets.

Le module adresse ip est composé des fonctions et procédures : Initialiser, Lire\_adresse, Convertir\_adresse, Compatible et  « >=».

*Initialiser* est une procédure quipermet d’initialiser une adresse à partir des valeurs entières de chaque octet.

*Lire\_adresse* est uneprocédure qui permet de lire une adresse dans un fichier texte et de l’initialiser en type adresse ip.

*Convertir\_adresse* est une fonction qui permet de convertir une adresse de type adresse ip vers une chaine de caractère avant de l’insérer dans le fichier des résultats.

*Compatible* est une fonction qui permet de déterminer si une adresse avec un masque est compatible avec une destination.

 «>= »  est une fonction permettant de