RAPPORT
PROGRAMMATION RÉSEAU
2019-2020

Auteurs :
DIAWARA Mohamed Lamine
XIA Eric

Introduction :

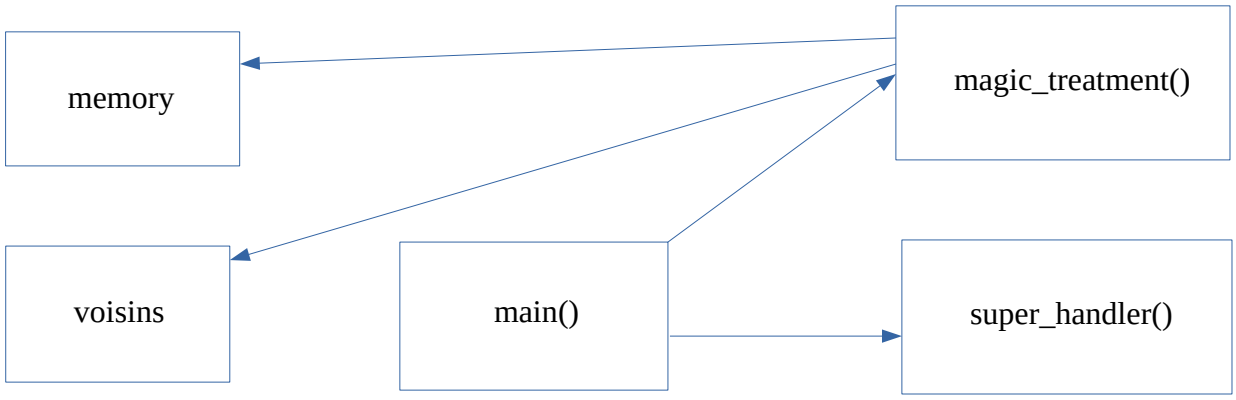
Dans le cadre de notre 3ème année en Licence Informatique parcours informatique générale à l'Université Paris Diderot nous avons eu pour tâche la réalisation d'un projet de programmation réseau.
Notre objectif était d'implémenter un dazibao (« journal à grandes lettres »), semblable à un « mur » de réseau social en utilisant le langage C#.

Ce rapport contient l'ensemble des éléments du projet. D'un point de vue technique tout d'abord, nous présenterons la structure du programme telle que nous l'avons imaginé, puis les spécifications plus détaillées qui en découlent. Nous décrirons le fonctionnement de notre projet dans son ensemble ainsi que les éléments qui prouvent le bon fonctionnement de celui-ci. Pour terminer la partie technique, nous présenterons nos impressions sur le projet concernant les difficultés techniques rencontrées et les perspectives ouvertes.

Sommaire :

I) STRUCTURE DU PROGRAMME :

Le programme est composé du [main()], de la memoire [struct memory] (aka table des données), les voisins [struct voisins], d'un parseur de tlv entrant [magic_treatment()] et d'un textuelle interactive [super_handler()].



STOCKAGE TRAITEMENT

1) Schéma des utilisations des modules :

Ce schéma montre que magic_treatment() fait tout sur les paquets entrants (et sortant sur demande de ces deniers). Donc une règle simple s'impose: les données dans memory et voisins sont toujours au format machine, et magic_treatment() fera le nécéssaire en conversion.

2) TECHNIQUES

Boucle à événements

C'est une boucle à évènement avec select() sur stdin et la socket.

Parseur de TLV

La fonction magic_treatment() parse les TLVs. Elle demande dabord à la fonction is_all_good_size() si le paquet entier est correcte, puis entre dans le parseur principale et execute.

Technique d'agrégation des TLV sortants

Beaucoup de type de TLV ont une fonction de la forme:
unsigned char * x_in_body(unsigned char * body, ...)

Ils ajoutent le TLV x et retournent la prochaine place pour un TLV dans body. (donc quasi-tout les TLVs sont ajoutable dans un même buffer (les node hash pourraient aussi être avec les autres mais si sa fait des erreurs, sa va être compliquer de trouver))

II) Fonctionnalités du projet :

Les extensions :
Vérification de la cohérence des nodes state :

Agrégation (partiel : par choix sur certains TLV) :

Calcul des hashes :

Les TLV node hash après un TLV network state request.
Les TLVs node state request est node state ensemble après un TLV node hash et node state request.

INTERFACE :

Textuelle interactive dans le terminal

Sa permet de changer le message (et afficher des choses comme la mémoire ou bien les voisins).

Difficultés rencontrées :

Les difficultés rencontrées ont été de deux ordres. Des difficultés techniques et des difficultés d'organisation.

Difficultés techniques

La réalisation de A à Z d'un réseau n'est parfois pas très aisée. On a néanmoins réussi à obtenir l'aide nécessaire, soit en rélisant plusieurs fois les cours, soit ailleurs en cherchant sur Internet.

Quant aux difficultés de programmation, elles sont venues principalement d'un bug d'une composante du compilateur avec plusieurs WARNING. Là, on a perdu un temps précieux. Finalement, on a pu trouver une solution adéquate.

Difficultés d'organisation

Dans ce projet, on a rencontré surtout des difficultés d'organisation. Ce n'est qu'au mois de mai que on a établi un certain équilibre entre nos différents projets et les autres tâches (les TPs). Cet équilibre a été très difficile à respecter, mais le travail a quand même avancé de manière significative.

Partie qui ne fonctionne pas

L'agrégation ne fonctionne que par paquet reçu (pas d'agrégation entre plusieurs paquets, donc les pairs doivent aussi faire des agrégations).

III) Parties complémentaires

DIFFERENCE / PAS DECRIT DANS LE PROJET:

Le premier TLV qu'on envoie au début (avant while(1)) est un TLV network state request. Sa permet de remplir rapidement la table des données puisque c'est de base vide.

Le pair courant n'a pas de données de base (technique secrète d'invisibilité). (On pourrait avoir un message de base, ou bien forcer un message, mais c'est méchant !, donc le pair courant aura un message quand il le voudra) (On peut considérer qu'un nouveau pair sans données est un espion)

En changeant de données : on envoie à tout les pairs un node state.

Quand le nombre de voisins est inférieur à 5, on envoie à tous un TLV neighbour request (pas à un seul)

A REMARQUER:

La table des données contient aussi celui du pair courant. La structure de memory contient un tableau [sens_tri], qui permet d'ajouter de nouvelles choses *incrémentalement* et seulement changer les indices dans ce tableau. ([sens_tri] est très utile, simple et efficace, on peut aussi faire des recherches par dichotomies mais les folds sont plus classes!)

Les folds sont implémentés pour les voisins et les données (puisque ce sont les stockages), avantage: c'est classe!, inconvénient : c'est pas efficace quand on a pas besoin de tout parcourir, c'est aussi un peu casse tête , et très compliqué quand il faut passer des structures comme void *.

Conclusion :

A l'aide de ce projet nous avons pu comprendre et expérimenter les différentes étapes de la conception du programmation réseau. De plus la programmation nous a permis d'améliorer nos connaissances de langage C#.

En plus d’être un projet pédagogique, il est aussi il est aussi ludique est nous a donné beaucoup de liberté dans le code et dans la conception.