Keppel Arnaud #b2o
Casanova Antonin #LaMafiaCorse

Projet Réseau

Calculateur réparti

Introduction

Durant notre UE de réseau, nous avons été amenés à faire un projet de calculateur réparti, en se basant sur les protocoles UDP TCP/IP V4 et V6.

Ainsi, nous avons mis en place un orchestrateur et des satellites de calcul, c'est à dire 5 programmes distincts, un orchestrateur ainsi que les 4 opérations de base + / * /.
Afin de mener à bien ce projet, nous avons utilisé diverses techniques de programmation, telles que la création de threads avec OpenMP, l'envoi de signaux pour terminer proprement l'exécution ainsi que tous les outils liés aux fonctions des sockets en C.

Nous allons par la suite détailler le fonctionnement global de notre projet.

Principes généraux de fonctionnement

Pour le fonctionnement global, 5 programmes sont exécutés simultanément, les 4 opérations de base ainsi que l'orchestrateur.

Concernant les protocoles utilisés, l'orchestrateur se base sur le protocole IPV6 mais assure également la portabilité IPV4. Le projet est compatible avec les deux protocoles, IPV4 et IPV6.

Pour commencer, nous allons détailler l'élément central, l'orchestrateur.

L'orchestrateur se charge de :

- Récupérer les saisies utilisateurs
- Voir si elles sont valides
- Interpréter ces données, puis les expédier par la suite
- Recevoir et afficher le résultat final
- Vérifier que chaque opérateur est en vie

Voyons maintenant les opérateurs de calculs :

- Recevoir le calcul initial
- L'analyser grâce à un parser
- Effectuer le calcul en soi
- Émettre la réponse à l'orchestrateur
- Donner un signe de vie toutes les X secondes

Mécanismes des fonctions principales

int envoi_string(char *adresse, char *port, char *data, int tr) ;

Cette fonction est une “fabrique à socket”. Elle va permettre d’envoyer via sendto la requête donnée en argument (data) à l’adresse et au port, puis le fd du socket est libéré.
La variable tr nous permet d’ajouter où non l’adresse IP ainsi que la date de renvoi du message (elle est égale à 1 exclusivement dans les satellites de calcul.)

char* reception_string(char *port) ;

Cette fonction nous permet de recevoir des données transmises sur un port par envoi_string. La fonction retourne le message réceptionné.

int parser(char *calcul) ;

Le parser détermine l’opérateur ainsi que les chiffres/nombres saisis (strtok), puis renvoie le calcul effectué. Elle est utilisée dans chacun des satellites.

void en_vie(char *c);

La fonction nous permet d’envoyer un signal de vie toutes les X secondes en envoyant un string à un port particulier de l’orchestrateur.

void en_vie_check(time_t* tab);

Tournant en boucle sur l’orchestrateur, elle assure que chacun des satellites a donné un signal de vie il y a moins de X secondes. Si cela n’est pas le cas, elle le désactive.

void boucle_emission(time_t* tab,int * check) ;

Cette fonction se charge de dispatcher aux satellites les différents calculs à l’aide d’un switch, tout en s’assurant que l’utilisateur n’utilise pas de satellites déconnectés ou occupés. Elle est elle aussi exécutée en boucle.

void boucle_reception(int* check) ;

Elle chargée d’afficher les calculs finaux émis par les satellites, tout en s’assurant que chaque calcul que l’utilisateur a saisi dispose d’une réponse.

Ces trois dernières fonctions sont exécutées en parallèle du programme orchestrateur, à l’aide des fonctions de openMP (https://en.wikipedia.org/wiki/OpenMP.)

Pour faciliter le développement du projet, nous avons créé un Makefile pour repartir sur des bases saines à chaque modification (routine de test = compilation en -Wextra + exécution.)
Nous avons également définis de nombreuses variables globales dans le header (snd_rcv.h) comme DELAI_VIE, DELAI_CALCUL et DELAI_MORT afin de pouvoir aisément modifier ces valeurs dans le futur.

Difficultés rencontrées

Il n'y a pas eu de grosses difficultés particulières concernant la structure du programme, il nous a néanmoins fallu une réflexion poussée afin de mettre en oeuvre l'ensemble des mécanismes demandés (gestion d'un satellite défaillant,gestion des timers, Openmp, exécution de plusieurs boucles while en parallèle, gestion des signaux, structures diverses et variées liées aux sockets et adresses,...)

La gestion des threads et de la mémoire partagée demande une certaine connaissance.

L'affichage de l'orchestrateur est particulièrement difficile. Il est parfois hasardeux, mais reste tout de même fonctionnel.

Conclusion

Ce projet est un condensé de nos diverses compétences acquises tout au long de notre licence
Il nous a permis d'implémenter pour la première fois un programme utilisant les protocoles de dialogue usuels (TCP/IP,UDP) et voir l'une de ses nombreuses applications.