Rapport de projet

Rappel du sujet :

On désire réaliser un système de calcul distribué, c'est-à-dire un système permettant de déterminer des valeurs $f_k(x_1,x_2...x_n)$, le calcul des f_k étant effectué sur des « nœuds » spécialisés. Le système doit permettre l'ajout ou le retrait de nœuds, voire la panne d'un nœud sans provoquer de perturbation s'il y a une redondance.

Liste des commandes :

Master :

quit : lance la procédure de terminaison générale
send symbl(arg1,arg2,....) : demande de résolution
etat : affiche les fonctions connues ainsi que les Slaves associés
help : affiche l'aide

Slave :

quit : lance la procédure de terminaison du Slave .

Type de message :

Message des Slaves:

Message de connexion au master:	[C Symbl Ip Port]
Message de déconnexion :	[D Id]
Message de réponse:	[R Id Réponse]
Message d'erreur:	[I Id Réponse]
Keep Alive	[K id]

Symbl : correspond aux symbole de l'opération (opération géré : +, -, /, *, %)

Message Master:

refus de connexion:	[E]
Acceptation de connexion:	[A]
Requête:	[Q arg1,arg2,arg3....)]
Terminaison master:	[S]

Pour les messages nous avons utilisé un format composé d'une lettre majuscule indiquant l'action à effectuer (le nombre d'action étant limité une seule lettre est suffisante pour traiter cette information), et le séparateur est représenté par un espace.
Lors d'une connexion d'un Slave au Master, une requête de connexion est effectué pour savoir si la connexion est possible. Cette requête contient le symbole de l'opération géré, ainsi que l'ip et le port du Slave. Le Master, lui attribue alors un Id unique afin de l'identifier plus facilement.

Lors d'une requête, le Master gère la gestion des Slaves disponible et adéquat. Ce qui nécéssitera que l'envoi de la suite d'arguments sous la forme d'une chaine de caractère débutant par le premier argument et finissant par une parenthèse fermante, chaque arguments étant séparé par une virgule. De son coté le Slave va effectuer l'opération si possible et renvoyer la réponse au Master.

Concernant la gestion des arguments envoyés, le Master va controler que la syntaxe de la demande soit correcte (vérification que les arguments soient bien des nombres), alors que les Slave vont vérifier si la résolution de l'opération est possible (division par 0 par exemple). Ceci ayant pour but d'alléger la charge du Master.

Détails des intéractions entre Slave et Master du point de vue des messages :

les messages sont représenté ici par leurs lettres Majuscule
Le Slave est représenté par la lettre S et le Master M
Le sens des messages sera représenté entre parenthèse de cette manières (S → M) ou (M → S)

		Message	Sens
1.	Si	C	(S → M)
2.	Alors	A ou E	(M → S)
3.	Si « A », Tout les x temps	K	(S → M)
4.	Si	Q	(M → S)
5.	alors	R ou I	(S → M)
6.	déconnexion/terminaison	D ou S	(S → M) ou (M → S)

Concernant le fonctionnement du programme voir les schéma ci-dessous:

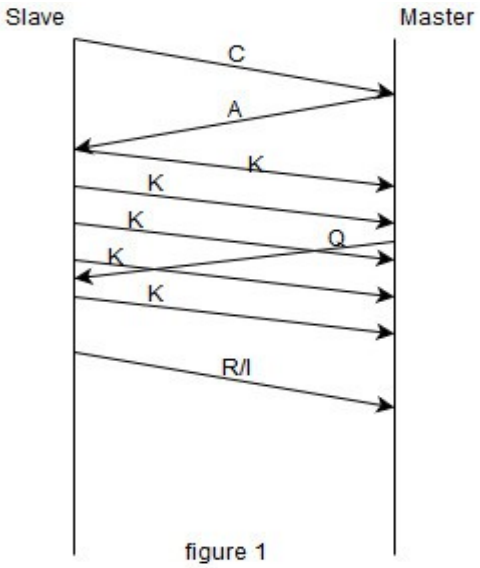

figure 1

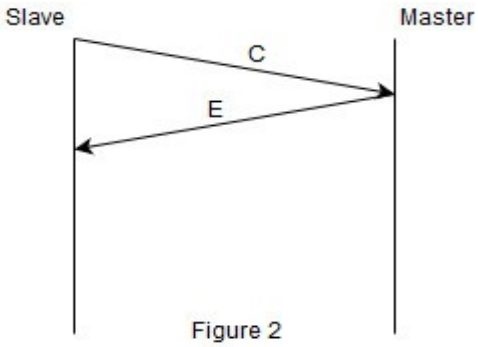

Figure 2

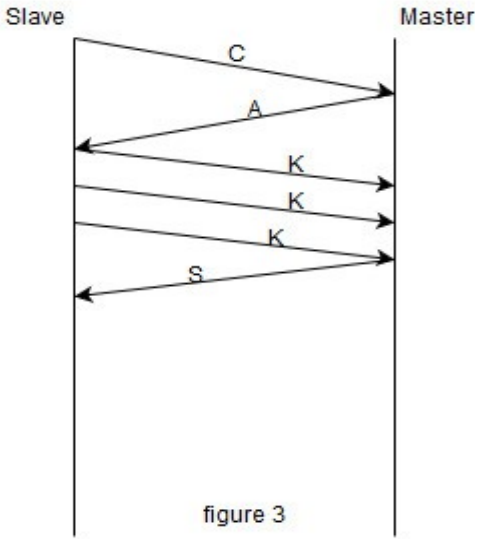

figure 3

Sur la Figure 1 :

Nous pouvons observer qu'un Slave tente une connexion au Master, ce dernier accepte la connexion et renvoi un message A.
A partir de ce moment le Slave envoi régulièrement un message de Keep Alive au Master et ce même si une requête est effectué.
Suite a l'envoi de la requete (message Q) le serveur répond par un message R ou I en fonction du bon fonctionnement de l'opération.

Sur la figure 2 :

Ici le Slave tente une connexion au master qui refuse immédiatement la connexion (message E)

Sur la figure 3 :

La connexion s'effectue correctement et le Slave envoi régulièrement des Keep Alive. Jusqu'a ce que le master se termine et envoi un message de terminaison au Slave.

Keep Alive :

Le Keep Alive est un message composé sous cette forme : [K Id] une fois receptionner par le Master, ce dernier va mettre à jour dans sa liste de Slave l'horaire auquel la reception du message a été effectué.
Si aucun Keep Alive n'est effectué après 10 secondes, le Janitor (partie du Master) va à son passage va mettre l'ID sur Slave a -1, la variable de keep Alive est mise à 0, les ports et IP sont mis à NULL. Si une requête était en cours, elle est redirigé si possible vers un autre Slave sinon elle est abandonnée.

Structure de donnée:

Structure du Master :

la structure servers représente un esclave, elle porte les informations permettant de joindre le serveur, son identifiant sur le réseau, son état ainsi que son travail en cours (s'il éxiste).

La structure fct_family est une liste chainé servant à regrouper les esclaves traitant le même type d'opération, elle porte le symbole associé à ces fonctions.

La structure Struct_for_listener est la structure principale commune à tout les thread du Master, c'est elle qui porte le mutex utilisé pour la synchronisation entre les thread (voir plus loin) , c'est également elle qui définie l'ID donnée a un esclave, elle est aussi composée des adresses d'écoute et de leurs adresses associées (Ipv4 / Ipv6), pour finir elle est composée de la structure « fct_family »

La structure Struct_for_janitor : cette structure est dédiée au thread janitor, elle est composé d'un pointeur vers la structure principale, et d'un mutex permettant au Janitor de savoir si le Master est en cours d'arrêt.

Structure Slave :

La structure Struct_for_listener_c étant la structure principale des esclaves, elle comporte un pointeur sur fonction pointant vers la fonction associé à l'esclave, le symbole,l'id donné par le Maitre, l'adresse du Maitre, ainsi que les informations utile pour la socket d'écoute.

La structure Struct_for_ka : Il s'agit de la structure nécéssaire au Keep Alive comportant le type d'Ip (Ipv4/Ipv6), son Id ainsi qu'un mutex de terminaison.

Granularité de la synchronisation des threads du Master:

La structure principale a été protégé à l'aide d'un mutex, permettant ainsi la sécurité des données, la synchronisation entre les thread et d'éviter l'accès simultané à la même ressource. Cette structure n'est alors accessible que par un unique thread à la fois, que ce soit les thread lié à l'écoute des sockets Ipv4 ou Ipv6 ou encore au janitor.

Le choix de ce mutex pose quelque soucis de performance notamment que l'accès a différente ressources dans le même temps est impossible. Mais son point fort est sa facilité de mise en place (un seul mutex à gérer).
De ce fait, on aurait pu rendre la granularité plus fine, en mettant en place un mutex sur chacune des famille de symbole ou pour aller plus loins sur chacun des serveurs connecté.

Terminaison des programmes :

Dans le cas d'une terminaison propre du Master, un message S est envoyé aux esclave à l'aide de la commande « quit », les sockets seront libérées, à l'aide des mutex le Keep Alive des Slave et le Janitor du Master seront prévenu de la terminaison du programme ce qui explique la latence qu'il peut y avoir car il sera nécéssaire d'attendre maximum un cycle de sleep avant que ce thread se termine.

Si un Slave a reçu la commande « quit » avant de finaliser une requête, cette dernière sera finalisé et la procédure de terminaison sera effectué une fois que la requête est terminée.

Dans le cas d'une terminaison propres d'un Slave le message D sera envoyé au master, dans ce cas là le janitor clean le serveur de la liste.

Limites d'implémentation :

-Les nombres négatif ne sont pas gérés (a l'envoi) → choix d'implémentation non significatif pour le déroulement du programme.

-Mutex sur structure manquant de précision → comme cité plus haut une amélioration peut être faites à ce sujet.

-Dans le cas d'une grosse latence sur un Keep Alive, permettant au Janitor de le considéré comme « mort », le Slave sera supprimé sans vérification, et à l'arriver du prochain Keep Alive le Master n'en tiendra pas compte, le Slave tournera alors dans le vide. → Peut être réglé en produisant l'arrêt du Slave si ce cas se produit.

-Le master ne vérifie pas les collision d'adresse ou de port, tout comme les esclaves ne vérifie pas s'ils peuvent s'approprié un port qui leurs est attribué ce qui provoque un comportement indéterminé. → peut être réglé en implémentant une vérification.

-Aucun test n'est effectué sur les fonctions de librairie ce qui peut provoquer des erreurs non prévue.