Ho Ngoc Kim Ngan

L3 Informatique

Université de Poitiers

RAPPORT PROJET

PROGRMMATION AVANCÉE EN C

FACULTÉ DES SCIENCES FONDAMENTALES ET APPLIQUÉES UNIVERSITÉ DE POITIERS

Décembre 2023

Table des matières

I-Liste des fichiers et leurs buts :

1. Fichiers principales :

client.c: implémenter d'un client qui communique avec un master à l'aide de tubes nommés et de sémaphores. Le client envoie des commandes au master en fonction des arguments passées en ligne de commande.

client_master.c: implémenter des fonctions liées à la gestion des sémaphores, utilisées probablement pour la synchronisation entre les processus du client et du maître. Ces fonctions sont utilisées pour créer, récupérer, entrer et sortir de sections critiques, ainsi que pour détruire les sémaphores.

client_master.h: définir les interactions entre client et master, définissant les ordres possibles, les réponses attendues, les noms des tubes, les sémaphores et des fonctions utilitaires associées à la gestion des sémaphores.

master.c: implémenter d'un master, recevant des ordres d'un client via des tubes nommés et communiquant avec des workers via des tubes anonymes pour effectuer diverses opérations sur un ensemble de données.

master_worker.c: implémenter des fonctions pour faciliter la communication entre le master et le worker, ainsi que une fonction pour créer un worker.

master_worker.h: définir les interactions entre le master et worker, définissant les ordres possibles, les réponses attendues et des fonctions utilitaires associées à la gestion de la communication entre master et worker et de la création d'un worker.

worker.c: immplémenter d'un worker, ce worker est destiné à être utilisé avec un programme principal (master) pour effectuer diverses opérations, telles que le calcul du minimum, du maximum, de la somme, etc... sur des éléments.

2. Fichiers fourni :

README : détaillassions de compilation et de test.

config.h : mode trace.

Makefile : compilation.

myassert.c : implémenter des fonctiosn pour la gestion des assertions dans le programme.

myassert.h : gérer les assertions conditionnelles dans le programme .

utils.h : définir des fonctions utilitaires éventuellement.
utils.c : implémenter des fonctions utilitaires éventuellement.

test_client.sh : le test pour le programme rmsempipe.sh : détruire des sémaphores et des tubes.

II- Les protocoles de communication :

1. Les tubes et les sémaphores utilisés pour la communication :

Client – master :

- clientToMaster et masterToCLient : tubes nommée.

- Un sémaphore pour synchroniser entre master et client, un sémaphore pour éviter que deux clients ne parlent simultanément. (le problème qui ne marche pas).

Master – workers :

- Utiliser les mêmes tubes nommées que le client pour communiquer.
- masterToFirstWorker[2] et firstWorkerToMaster[2] : tubes antonymes.
- workerToMaster : tubes anonyme pour recevoir le répondre des worker to master.

Worker et son père:

- workerToParent et parentToworker: communication avec le père.
- workerToMaster : communication avec le master (un tube en écriture).
- workerToLeftChild[2] et LeftChildToWorker[2] : communication avec le fils gauche s'il existe.
- WorkerToRightChild[2] et rightChildToMaster[2] : communication avec le fils droit s'il existe.

2.Les ordres :

Ordre de fin :

- Le client envoie un ordre de fin au master.
- Le master envoie l'ordre de fin au premier worker (s'il existe).
- Si le worker a des fils(gauche ou droit), il envoie l'ordre de fin à chacun d'entre eux récursivement.
- Chaque worker reçoit l'ordre et effectue les opérations de clôture nécessaires.
- Les workers confirment la réception au son père.
- Le master attend la confirmation du premier worker, et envoie l'affirmation au client.

Ordre howmany :

- Le client envoie l'ordre de cardinalité au master.
- Si le premier worker n'existe pas, l'ensemble est vide.
- Sinon le master envoie l'ordre de cardinalité au premier worker.
- Si le worker a des fils(gauche ou droit), il envoie l'ordre de cardinalité à chacun d'entre eux récursivement.
- Chaque worker reçoit l'ordre et effectue les opérations de calcul nécessaires.
- Les workers confirment la réception au son père avec le résultat à la suite.
- Le master attend la confirmation du premier worker, envoie l'affirmation et le résultat au client.

Ordre min :

- Le client envoie l'ordre min au master.
- Si le premier worker n'existe pas, l'ensemble est vide, le master envoie l 'affirmation l'ensemble est vide au client.
- Sinon le master envoie l'ordre min au premier worker.
- Si le worker a des fils(gauche ou droit), il envoie l'ordre à chacun d'entre eux récursivement.
- Chaque worker reçoit l'ordre et effectue les opérations nécessaires.
- Les workers confirment la réception au son père avec le l'élément plus petite à la suite.
- Le master attend la confirmation du premier worker, envoie l'affirmation et le résultat au client.

Ordre max :

Même principale que l'ordre min.

Ordre d'existence :

- Le client envoie l'ordre d'existence au master.
- Si le premier worker n'existe pas, l'ensemble est vide, le master envoie l'affirmation négative au client.
- Sinon le master envoie l'ordre d'existence au premier worker.
- Si le worker a des fils(gauche ou droit), il envoie l'ordre à chacun d'entre eux récursivement.
- Chaque worker reçoit l'ordre et effectue les opérations nécessaires.
- Les workers confirment la réception au son père avec l'élément et sa cardinalité à la suite.
- Le master attend la confirmation du premier worker, envoie l'affirmation et le résultat au client.

Ordre de somme :

- Le client envoie l'ordre de somme au master.
- Si le premier worker n'existe pas, le master envoie l'affirmation au client avec la somme est égale à 0.
- Sinon le master envoie l'ordre de somme au premier worker.
- Si le worker a des fils(gauche ou droit), il envoie l'ordre à chacun d'entre eux récursivement.
- Chaque worker reçoit l'ordre et effectue les opérations de calcul nécessaires.
- Les workers confirment la réception au son père avec le résultat à la suite.
- Le master attend la confirmation du premier worker, envoie l'affirmation et la somme au client.

Ordre d'inserttion :

- Le client envoie l'élément à insérer au master.
- Si le premier worker n'existe pas, on crées le premier worker avec l 'élément reçu du client.
- Sinon le master envoie l'ordre d'insertion au premier worker.
- Si le worker a des fils(gauche ou droit), il envoie l'ordre d'insertion à chacun d'entre eux récursivement.
- Chaque worker reçoit l'ordre et effectue les opérations nécessaires.
- Les workers confirment la réception au son père.
- Le master attend la confirmation du premier worker, envoie l'affirmation et au client.

Ordre s'insertion d'un tableau d'éléments :

- Le client envoie le tableau d'élément à insérer au master.
- Pour chaque l'élément du tableau, on effectues l'insertion selon l'algorithme de l'ordre d'insertion.
- Le master attend la confirmation du premier worker, et envoie l'affirmation et le résultat au client.

Ordre d'affichage :

- Le client envoie l'ordre d'affichage au master.
- Si le premier worker n'existe pas, l'ensemble est vide.
- Sinon le master envoie l'ordre d'affichage au premier worker.

- Si le worker a des fils(gauche ou droit), il envoie l'ordre à chacun d'entre eux récursivement.
- Chaque worker reçoit l'ordre et effectue les opérations de calcul nécessaires.
- Les workers confirment la réception au son père et afficher son élément avec sa cardinalité.
- Le master attend la confirmation du premier worker, envoie l'affirmation au client.

III- Les problèmes :

Dans le processus de réalisation du projet, j'ai fait face à plusieurs difficultés. Bien que j'aie déployé des efforts pour les résoudre, certaines persistent, représentant ainsi des défis non encore surmontés. Voici un aperçu des problèmes rencontrés :

Le sémaphore :

Initialement, j'ai pris en compte la nécessité d'avoir deux sémaphores, l'un pour la communication entre le client et le master, et l'autre pour éviter que deux clients ne parlent simultanément.

Cependant, dès l'ajout des sémaphores dans mon programme, celui-ci a cessé de fonctionner correctement. Pour surmonter cette situation, j'ai commenté les lignes de code liées aux sémaphores et ajouté un délai (sleep(1) dans la boucle **loop**) pour poursuivre le codage du programme.

L'ordre d'insertion d'un tableau :

Lors de l'exécution de la commande d'insertion d'un tableau, un message d'erreur indiquant l'impossibilité de recevoir le tableau a été affiché. Cependant, l'insertion individuelle de chaque worker semble fonctionner correctement.

Avant de tester la commande d'insertion, toutes les autres commandes fonctionnaient normalement. Cependant, après avoir lancé la commande d'insertion, des erreurs sont survenues lors de l'exécution des commandes de somme et d'existence.

```
[master] début
   Master a reçu l'ordre du client
[master] ordre insertion
    [worker (5223, 5220) {10}] : début worker
[master] fin ordre
   Master a reçu l'ordre du client
[master] ordre existence
/--------------------------
| Erreur détectée !
|       fichier  : /home/kimngan/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src/master.c
|       ligne    : 347
|       fonction : orderExist
|       pid      : 5220
|    Message :
|       -> ne pas pouvoir lire l'accusé de réception du premier worker
|    Message systeme:
|       -> Success
|    On stoppe le programme
\--------------------------
       Worker a reçu l'ordre
    [worker (5223, 5220) {10}] : ordre exist
    [worker (5223, 5220) {10}] : fin ordre
```

Impact sur la vérification du résultat :

En raison des problèmes liés à l'ordre d'insertion d'un tableau, je suis actuellement dans l'incapacité de vérifier le résultat lorsque des workers sont présents dans l'ensemble.

Notamment, si l'ensemble est vide, le résultat est conforme. Cependant, la présence de workers complique la vérification du résultat.

Ces défis persistent malgré mes tentatives antérieures de résolution, et je continue à explorer des solutions afin de surmonter ces obstacles dans le cadre de la réalisation du projet.

```
kimngan@kngan:~/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src$ ./client howmany
nombre d'élement: 0
nombre d'element disttinc: 0
kimngan@kngan:~/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src$ ./client sum
Sum: 0.000000
kimngan@kngan:~/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src$ ./client min
L'ensemble est vide
kimngan@kngan:~/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src$ ./client max
L'ensemble est vide
kimngan@kngan:~/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src$ ./client exist 10
L'element 10.000000 n'existe pas
kimngan@kngan:~/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src$ ./client local 3 5 20 0 10
[2 8 1 2 8 6 0 6 6 5 1 1 2 4 7 3 9 4 6 4]
Elément 5 présent 1 fois (1 attendu)
=> ok ! le résultat calculé par les threads est correct
kimngan@kngan:~/Bureau/L3/Projet_C/CODE_FOURNI/PROJET/src$ ./client stop
STOP_OK
```