Lucas Vadé Grelet
Leonard Duhem

Licence Informatique L3 - 2025-2026
UFR SFA – Université de Poitiers

Projet C avancé

Sommaire :

Explication du projet :

Explication global :

L'objectif du projet est de réaliser un système permettant de déterminer si un nombre est premier .
Ce système repose sur trois types de programmes : client, master et worker(s)
Le fonctionnement est inspiré du crible d'Ératosthène mais sous sa forme pipeline, appelée crible de Hoare l'idée principale est d'avoir, un master qui coordonne l'ensemble, des clients indépendants qui envoient des requêtes, un ensemble de workers organisés en pipeline, chacun chargé, d'éliminer les multiples d'un nombre premier
Le master interroge ce pipeline pour répondre aux requêtes de primalité
en résumé : CLIENTS → MASTER → PIPELINE DE WORKERS → RÉPONSE

Fichier principaux :

Client.c :

BUT DU FICHIER : Le programme "client.c" représente l'interface entre l'utilisateur et le système de calcul (master + worker)
Il interprète les ordres passés en ligne de commande et envoie ces ordres au master via des fifo synchronisé par sémaphore IPC
Il implémente également un mode local, demandé dans la partie 3,3bis qui exécute un crible multi-threadé indépendant du master

RÔLE GÉNÉRALE :
- Interprète les arguments envoyés par l'utilisateur
- Se synchronise avec le master grâce aux sémaphores IPC
- Utilise les fifo crées par le master pour transmettre les ordres et recevoir les résultats
- Propose un mode spécial pour afficher les workers ouvert et un mode local indépendant du master.c

Interprétation des arguments d'entrée : parseargs() vérifie les arguments et retourne l'ordre parmi : ORDER_COMPUTE_PRIME, ORDER_HOW_MANY_PRIME, ORDER_HIGHEST_PRIME, ORDER_STOP, ORDER_COMPUTE_PRIME_LOCAL
En cas d'erreur, un message est affiché puis usage() termine le programme

Communication avec le master : Le client récupère les sémaphores sem_mutex et sem_sync
Il entre en section critique avec p(sem_mutex)
Il écrit l'ordre (et éventuellement le nombre) dans le fifo client -> master
Il lit la réponse dans le fifo master -> client
Il libère la section critique et réveille le master via v(sem_sync)
Toutes les interactions passent uniquement via deux fifo : client_to_master.fifo et master_to_client.fifo

Mode local : Le client exécute un crible d'Ératosthène multi-threadé
Tableau tab[2…n] initialisé a true, nombre de thread : sqrt(n)
Chaque thread élimine les multiples de son diviseur, un mutex global protège l'accès concurrent au tableau, affichage final des nombres premier trouvés
Ce mode ne communique ni avec le master ni avec les fifo

Mode spécial showworker :
execute « ps -C worker.o -o stat,cmd | grep -E 'R|S|D|I' »
Affichage des workers en cours d'exécution. Aucune communication avec le master.c

Master.c

But du fichier : Le fichier master.c implémente le processus maître, il coordonne l'ensemble du système : création des fifo et sémaphore, lance du premier worker, construction dynamique du pipeline de Hoare, gestion des requêtes client.c, supervision du pipeline (arrêts, re-tests, propagation du stop)

Fonctionnalité principale : Le master : crée les fifo client_to_master et master_to_client, crée deux sémaphores : sem_mutex (protège l'accès au fifo) et sem_sync (synchronise fin du dialogue client)
Crée deux tubes anonymes pour le pipeline : pipeMW (master -> worker) et pipeWM (worker -> master)
Lance le premier worker chargé du nombre premier 2, le premier worker envoie immédiatement 2 au master pour initialiser : highest_prime = 2 ; nb_primes =1 ; last_tested = 2

Boucle principale : loop() attend les clients en permanence : lit l'ordre envoyé dans le fifo, si nécessaire, lit le nombre, traite la requête, renvoie le résultat au client, attend que le client ait fini de lire (grâce a p(sem_sync))

Traitement des ordres :
compute : le nombre est envoyé dans le pipeline, chaque worker test et transmet au suivant, le master lit un unique message : >0 → le nombre est premier ou = 0 → le nombre n'est pas premier
Il met a jour last_tested, highest_prime et nb_primes
how_many : renvoie nb_primes highest : renvoie highest_prime,
stop : renvoie un ACK au client, propage -1 dans le pipeline et termine en supprimant fifo + sémaphore

worker.c

But du fichier : le programme worker.c implémente un maillon du pipeline de Hoare
Chaque worker représente un nombre premier, et filtre les multiples successifs
Le pipe est dynamique : les workers se créent au fur et à mesure

Fonctionnement interne
Lors d'un execv, chaque worker reçoit : argv[1] = fdRead (entrée)
argv[2] = fdWriteMaster (sortie vers master) argv[3] = myPrime (premier géré par ce worker)

Boucle principale (fonction loop)
à chaque nombre n reçu :
Si n==-1 : transmet stop au prochain worker s'il existe, attend sa terminaison, termine
si n==myPrime : c'est un succès, renvoie myPrime au master
si n divisible par myPrime : renvoie 0 au master (ce n'est pas un premier)
sinon, transmettre à un worker suivant : si ce worker n'existe pas encore, création d'un tube, fork d'un nouveau worker (premier = n), mémorisation du PID, ouverture du tube
sinon : on transmet simplement dans le tube du suivant
ce mécanisme réalise automatiquement le crible, chaque worker filtre une couche et crée le prochain niveau
fin de vie : A la fin, ferme ses tubes, attend éventuellement son successeur, affiche un message de terminaison

Fichier de module :

master_client.h, master_client.c

Ils regroupent tout ce qui est commun au master et au client : définitions des constantes d'ordres, noms des FIFO, gestion des sémaphores et des tubes nommés. On y trouve aussi les fonctions d'affichage des résultats côté client, l'envoi des ordres, ainsi que l'implémentation du mode local multithreadé.

master_worker.h, master_worker.c

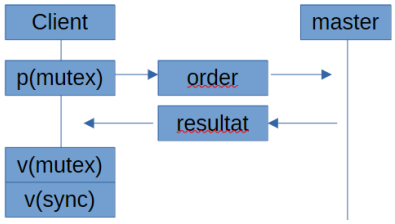
Ils forment un petit module dédié à l'interface entre le master et les workers. Ils isolent les éléments propres au couple master/worker et permettent de séparer clairement les rôles : le comportement détaillé des workers reste dans worker.c, tandis que master_worker.h définit l'interface associée.

Protocole

Le projet repose sur une architecture en trois niveaux : **client → master → pipeline de worker**
Chacun communique selon la méthode imposé par le sujet : **client ↔ master** et **master ↔ workers**

1) client ↔ master

communication bidirectionnelle : client_to_master.fifo et master_to_client.fifo
chaque échange suit un cycle précis :
- le client entre en section critique : p(sem_mutex) → garantit qu'un seul client parle au master a la fois
- le client écrit l'ordre dans client_to_master (si besoin, il écrit un nombre n)
- le master lit l'ordre dans le fifo
- le master traite la requête : **compute** → dialogue avec pipeline, **how_many** → renvoie nb_primes
  highest → renvoie highest_prime, **stop** → prépare la fermeture du système
- le master écrit la réponse dans master_client
- le client lit la réponse, ferme le fifo,
- le client libère la section critique : v(sem_mutex) et reveille le master v(sem_sync)



résumé :
- le client attend toujours une réponse du master
- multiples clients autorisés, grâce au sémaphore sem_mutex
- le master attend la libération du client via sem_sync avant de continuer
- le protocole est déterministe : chaque ordre produit exactement une réponse

2) master ↔ pipeline des workers

pipeline : worker(2) → worker(3) → worker(5) → .....
le master communique uniquement avec : le premier worker via pipeMW(master→worker)
le dernier worker actif via pipeWM(worker→master)
les workers communiquent entre eux via des pipes crées dynamiquement
Quand le master reçoit COMPUTE n : le master écrit n dans le pipe pipeMW → worker(2)
chaque worker reçoit n et décide :
cas 1 : stop → n == -1→ propage -1 → termine
cas 2 : sucés → n == myPrime → renvoie myPrime dans pipeWM (n est premier)
cas 3 : divisible : n % myPrime == 0 → renvoie 0 dans pipeWM (n est pas premier)
cas 4 : le worker transmet n au worker suivant, si il y en a un il transmet, sinon il en crée un nouveau
Exemple :
pour 35 : master → 35 [w2] –non div-- [w3] –non div-- [w5] –div→ fail =0 → master
pour 41 : master → 41 → [w2] → [w3] → [w5] → [w7] → [w11] (création) → success=41 → master

protocole de fin (stop)
quand un client envoie l'ordre stop
le master renvoie d'abord un ACK au client
le master écrit -1 dans pipeMW
chaque worker : reçoit -1, le transmet a son successeur, ferme ses pipes, attend son fils, se termine
le pipeline se détruit proprement de gauche a droite

résumé communication :

Couche	Moyen	Type	But
Client → master	fifo	Synchrone	Envoyé ordre
Master→ client	fifo	Synchrone	Renvoyé résultat
Master→worker	Pipe	Asynchrone	Envoyer nombre a tester
Worker→ master	Pipe	Synchrone	Renvoyer résultat
Worker→ worker	Pipes dynamiques	Asynchrone	Propager le crible