le cnam
Occitanie

CONSERVATOIRE NATIONAL DES ARTS ET METIERS
CENTRE REGIONAL DE TOULOUSE

---

par

MILAZZO Christopher

Devoir sur les pipes en C

11/07/2020

ARBORESCENCE DU PROJET:

Le projet à été découpé comme suit :

- **main.c** => C'est le point d'entrée de l'application. Demande les infos à l'utilisateur, démarre les worker, cré les pipes puis lance la boucle principale du programme.
- **master.c** => Contient la boucle principale du master. Le master gère le pool de worker puis affiche le résultat de la recherche avant de supprime tous les FIFOs.
- **worker.c** => Contient la boucle principale d'un worker. Les workers cherchent les nombres premiers dans un sous-intervalle fournit par le master via un tube nommé « fifo_n » (voir schéma plus bas) puis retourne tous les nombre premiers trouvés via le descripteur de fichier « master_in_fd » (voir schéma).
- **mutils.c** => Contient divers méthodes utilisées par les procédures principales (worker.c et master.c). Le but de ce fichier est de découper l'application pour avoir un code moins laborieux à lire.

LE CHOIX DE L'ALGORITHME :

Le crible d'Ératosthène n'est pas compatible pour cet exercice, du fait qu'il ne soit pas parallélisable. En effet, ce dernier fonctionne de manière itérative ; L'itération N est dépendante du résultat de l'itération N-1.

Le plus efficace serait d'utiliser le principe de primalité de Fermat ; Le petit théorème de Fermat dit « Si p est un nombre premier et si a est un entier non divisible par p, alors $a^{p-1}-1$ est un multiple de p »
soit : $a^{p-1} \equiv 1[p]$
Cela se traduit par N = $2^{p-1}-1$ et si N mod p = 0 alors p est premier puisque p vérifie le petit théorème de Fermat.
Il faut y ajouter une variante pour traiter le cas des nombres pseudo-premiers :
M = $3^{n-1}-1$ si M mod n = 0 ET N mod n = 0 alors n est premier.

Cet algorithme ne nécessite que 6 opérations arithmétiques et 2 opérations logiques, ce qui est rapide pour un ordinateur mais l'inconvénient c'est qu'il faut manipuler des nombres extrêmement grand à cause des puissances. Au delà de 2 exposant 200 on a un débordement, ce qui oblige à devoir implémenter ou utiliser une lib capable de manipuler des nombres très grands.
Dans le cadre de ce devoir, vue que ce n'est pas le sujet et que je n'ai malheureusement pas assez de temps pour chercher et comprendre l'utilisation d'une telle librairie, je me suis contenté d'un algorithme simple avec néanmoins un peut d'optimisation :

```
n = nombre à tester
si n mod 2 = 0 ET n > 2 alors retourner FAUX //un nombre paire n'est pas premier car divisible 2
pour a de 3 à √n   // on ne teste qu'avec les nombres impaires car un nombre paire ne  divise pas
                      les nombre impaires
        si n mod a = 0 alors
                retourner FAUX // n n'est pas premier
        fin si
        a = a + 2
fin pour
retourner VRAI // n est premier
```

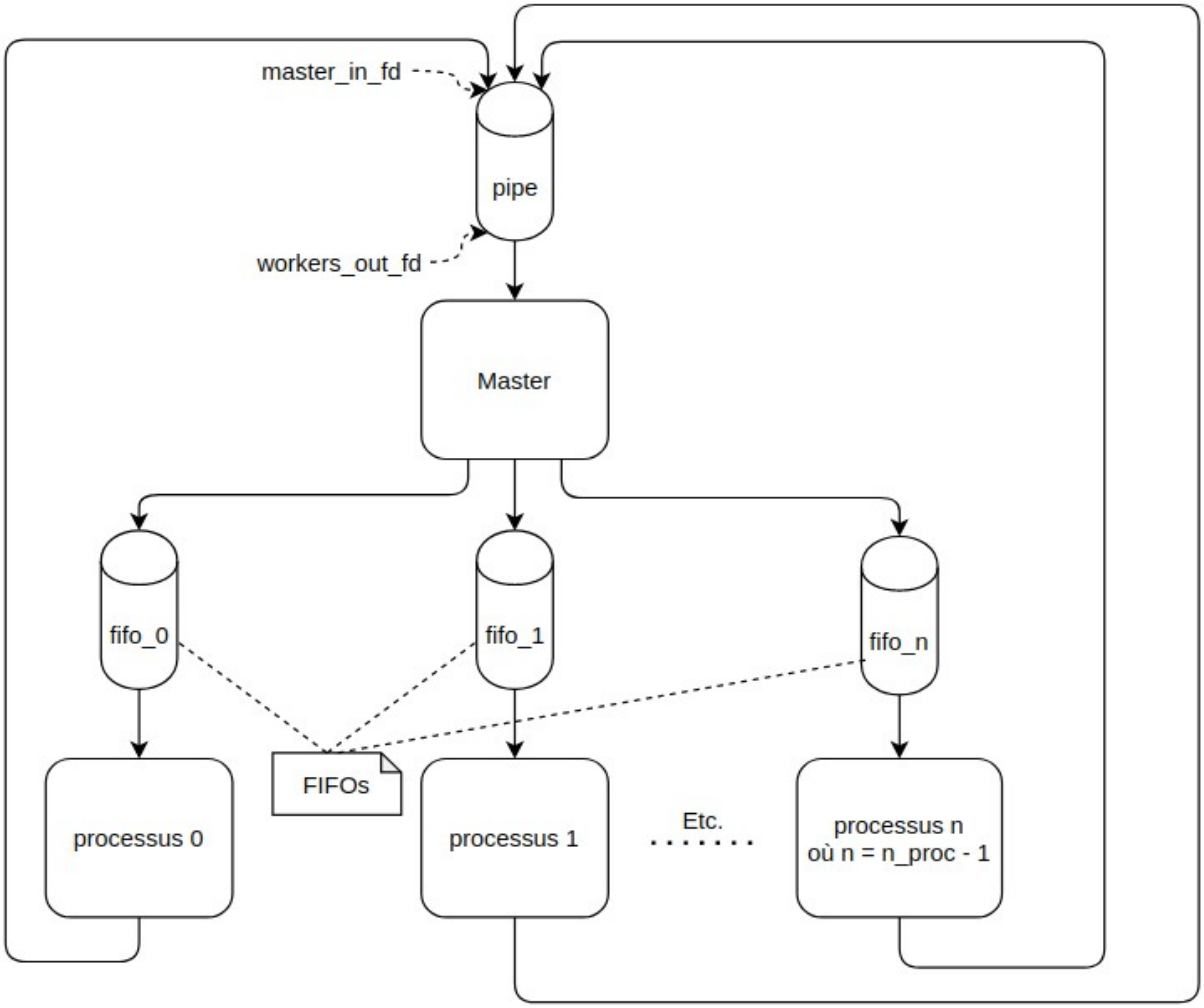
SCHÉMA ET PRINCIPE DE FONCTIONNEMENT :

Master gère un pool de worker (processus) au nombre de n_proc. L'utilisateur entre un nombre N > 2 puis un nombre de processus n_proc > 0. Ensuite, le programme commence à chercher tous les nombres premier compris dans [2 – N].

Pour cela, master cré n_proc * processus fils appelé des worker puis découpe l'intervalle de recherche en plusieurs petites sous-intervalles.
Le master ordonne aux worker de chercher les nombres premiers dans un sous-intervalle.
Dés qu'un worker trouve un nombre premier il averti le master via le pipe, puis continu jusqu'à la fin du sous-intervalle. Une fois terminé, il préviens le master qui lui fourni un sous-intervalle encore inexploré et ce jusqu'à ce qu'il n'y ai pus de sous-intervalle.

La communication dans les pipes se fait sous forme de chaines de caractères. A l'arrivée afin d'interpréter le message les modules utilisent la méthode **strsplit** implémentée dans **mutils.c** qui permet de découper le message en fonction des espaces.

Une fois tout l'intervalle couverts, le master attend que tous les worker ai terminés, avant de les détruire puis d'afficher les nombres premier classés dans l'ordre croissant.



COMPILATION :

gcc -o cmilazzo_devoir_tube main.c worker.c master.c mutils.c -lm
Je développe sur la distribution Ubuntu 18.04.4 LTS.
Processeur Intel Core i5-2320 CPU @ 3.00GHz * 4

NOTE : J'ai mis le binaire compilé « cmilazzo_devoir_tube » au cas où la compilation ne marcherait pas sur votre système.

UTILISATION :

Syntaxe :
$ ./cmilazzo_devoir_tube [-v]
L'option -v n'est pas obligatoire, elle permet juste de passer le programme en mode verbose.

Une fois démarrer le programme vous demande d'entrer une valeur entière N > 2 puis il vous demande d'entrer le nombre (un entier) de processus (worker) n_proc > 0.
Si n_proc > N alors n_proc = N (inutile d'avoir plus de worker que de travail à fournir).
Puis il cherche et affiche les nombres premiers compris dans l'intervalle [2 – N].

OBSERVATION :

Je n'ai pas eu besoins de gérer la synchronisation entre l'écriture des données par les worker et la lecture des données par le master. Les tubes semblent bloquer la lecture tant que l'on écrit dedans, ou alors j'ai eu énormément de chance, car je n'ai pas eu de problème lors de la trentaine d'essais sur une recherche dans un intervalle [2 – 50 000] avec 250 workers.

Le fichier **master.c** mériterais encore un peut de refactoring afin de le clarifier un peut mais je manque de temps. Malgré cela je suis satisfait de mon travail.