Projet ProgC Licence 3

Mathis DEVIGNE
— Mar, 17 mars, 2021 17:25

Vous pouvez voir ce fichier plus proprement en HTML à l'adresse :
https://hackmd.io/@mdevigne/BkZh3vcUi

Sommaire

Organisation du code

- myassert

myassert.c

Le fichier myassert.c sert à utiliser une fonction myassert donnée par le professeur pour s'assurer d'un bon retour. En nettoyant mon code, j'ai ensuite mis dans ce fichier des fonctions basiques et leur test de retour pour éviter une duplication de code.

Exemple avec mymkfifo:

```
void mymkfifo(const char *name, const int n){
    int retmkf = mkfifo(name, n);
    myassert(retmkf == 0, "Creating pipe");
}
```

myassert.h

Je les ai aussi définis dans le fichier myassert.h, pour qu'il soit accessible aux autres .c.

```
31  void myexecv(char *name, char **arg);
32  void mypipe(int t[]);
33  void myunlink(const char *p);
34  void mysemctlwithval(const int semid, const int semno, const int cmd, const int val);
35  void mysemctlnoval(const int semid, const int semno, const int cmd);
36  int mysemget(const int key, const int nb, const int order);
37  void mymkfifo(const char *name, const int n);
38  int myopen(const char * name, const int order);
39  void myclose(const int p);
40  void myread(const int fd, void *buf, const size_t size);
41  void mywrite(const int fd, const void *buf, const size_t size);
```

- master_client

Ces fichiers servent pour tous ce qui est utilisé par le master et le client.

master_client.c

J'y ai rajouté une fonction *mysemop* qui construis la structure *sembuf* puis la passe en paramètre d'un appel *semop* (cette fonction par du principe qu'il n'y a qu'un sémpahore).

master_client.h

Contient les définitions des clés de sémaphore, des tubes nommés, des ordres entre client et master.

- master_worker

Ces fichiers servent pour tous ce qui est utilisé par le master et le worker.

master_worker.c

J'ai rajouté une fonction *createWorker* qui créer un nouveau worker et renvoie une struct avec les informations pour communiquer avec lui. Cette fonction utilise une autre *prepArgWorker* qui renvoie les arguments du exec sous la forme d'une matrice de char.

master_worker.h

Contient la définition de la structure, de la fonction *createWorker* et des ordres entre worker et master.

```
typedef struct{
    int readp;
    int writep;
} dataCreateWorker;
```

- client.c

Ce fichier contient une partie local avec un crible d'Érathostène, et une partie en communication avec le master.
Un client peut demander au master si un nombre est premier, quel est le nombre premier le plus grand qu'il a déjà calculé, combien il en a calculé ou de se stopper.

- master.c

Lorsqu'on lance le master, il créer les tubes nommés et les sémaphores, qu'il initialise, pour le client. Ensuite il créer le premier worker puis passe dans une boucle ou il traitera les demandes de différents clients.

- worker.c

Un worker est créé pour chaque nombre premier traité. Une fois créer, il attend un nombre a traiter envoyé par son créateur. Si il ne peut pas savoir si le nombre est premier, il envoie le nombre au worker suivant, qu'il créer si non existant. Il envoie ensuite le résultat à son créateur qui l'envera jusqu'au master.

Protocoles de communication

Dans les fichiers *client.c*, *master.c* et *worker.c*, j'ai créer des structures pour stocker toutes les données utiles des fichiers, comme les tubes, les sémaphores ou les nombres à traiter.

Dans l'ordre : *master.c*, *worker.c*, *client.c*

```c
typedef struct{
    int semsid;
    int semtid;

    int pReadClient;
    int pWriteClient;

    int pReadWorker;
    int pWriteWorker;

    int maxN;
    int howManyPrime;
    int highestPrime;
} dataM;
```

```c
typedef struct{
    int p;
    int pWriteMaster;
    int pReadMaster;
} dataW;
```

```c
typedef struct{
    int order;
    int n;

    int semtid;
    int semsid;

    int pReadMaster;
    int pWriteMaster;
} dataC;
```

Communication master-client :

Le master et le client communique à travers deux tubes nommés créé par le master. Ces tubes sont ouverts par chaque client et à chaque itération du master.
De plus, le master créer deux sémaphores qu'il initialise. Un pour sécuriser l'ouverture des tubes entre clients, et un pour que le master attendent le client.

Communication master/worker-worker :

Lorsqu'un programme créer un worker, il créer deux tubes anonymes, un du créateur au worker, et l'autre du worker au créateur. Ces tubes sont envoyé en retour au créateur avec une structure *dataCreateWorker* et en paramètre d'exec du nouveau worker.