Faculté des Sciences
Département d'Informatique

INFO-F-201 – Systèmes d'exploitation

Projet de programmation système

TinyDB

L'ULB veut rénover son système de base de données d'étudiants et fait donc appel à vos compétences en programmation système pour reprendre un ancien projet de base de données laissé à l'abandon depuis plusieurs années. Au vu du nombre croissant d'étudiants, la base de données doit être particulièrement rapide et performante, exécutant de ce fait plusieurs requêtes simultanément.

1 Détails du projet

Ce projet est à réaliser par groupe de deux ou trois étudiants et comporte deux parties indépendantes
- la base de données implémentée en C
- des outils de monitoring en bash

Contactez Yannick Molinghen avant le 18 octobre 2022 si vous n'avez pas de groupe. Vous serez pénalisés si vous réalisez le projet seul sans en avoir discuté auparavant.

Vous êtes autorisés à utiliser des fonctionnalités de C++ telles que `vector`, `string`, `hashmap`, `new` et `delete`, les classes, etc[1].

Notez que ce premier projet servira de base pour le second[2], nous vous conseillons donc de penser à écrire du code maintenable.

2 La base de données

Cette section détaille le fonctionnement de votre programme de base de données dont l'exécutable doit s'appeler `tinydb`.

2.1 La table d'étudiants

Une base de données relationnelle se compose habituellement de différentes tables liées entre elles par des relations. Dans ce cas-ci, il n'y a qu'une table pour les étudiants, décrite dans la Table 1.

student		
Champ	Type	Signification
id	`unsigned int`	Identifiant unique
fname	`char[64]`	Prénom
lname	`char[64]`	Nom
section	`char[64]`	Section (informatique, médecine, …)
birthdate	`struct tm`	Date de naissance

TABLE 1 – La table d'étudiants avec le type de chaque champ.

1. Ce projet évalue les compétences en programmation système, vous ne serez donc pas récompensés pour avoir implémenté une magnifique chaîne d'héritage car ce n'est pas le sujet de ce cours.
2. Nous vous fournirons une solution type.

NB : le header `student.h` définit déjà une structure de type `student_t` qui correspond à cette table.

2.2 Format des requêtes

Vous devez gérer quatre types de requêtes pour lesquels vous trouverez des exemples dans le répertoire `tests/queries`. Vous pouvez ignorer les requêtes mal formées en affichant un message d'erreur mais sans arrêter le programme.

- `select <champ>=<valeur>`. Cette requête renvoie la liste des étudiants qui correspondent à l'unique filtre `<champ>=<valeur>`.
- `insert <fname> <lname> <id> <section> <birthdate>`. Cette requête insère un nouvel étudiant dans la base de données en vérifiant que l'ID n'existe pas déjà. Si l'id existe déjà, l'insertion échoue.
- `delete <champ>=<valeur>`. Cette requête supprime tous les étudiants qui correspondent au filtre donné.
- `update <filtre>=<valeur> set <champ_modifie>=<valeur_modifiee>`. Cette requête modifie tous les étudiants correspondant au filtre `<filtre>=<valeur>`, en donnant la valeur `<valeur_modifiee>` au champ `<champ_modifie>`.
- `transaction`. Ce mot-clef déclare le début d'une nouvelle transaction. Plus de détails sont donnés dans la Section 4.2.

2.3 Lancement de la base de données

La base de données doit se lancer via l'exécutable '`tinydb`' et prend en paramètre **obligatoire** le chemin vers la base de données. Au lancement, le contenu du fichier donné en paramètre doit être chargé à l'aide de la fonction `db_load` qui vous est fournie. Le programme se lance donc comme indiqué ci-dessous.

```
$ ./tinydb <chemin_vers_la_db>
```

2.4 Comportement de la base de données

Une fois lancée, la base de données se présente sous forme d'une ligne de commande qui accepte des requêtes (décrites dans la Section 2.2) sur stdin.

```
$ ./tinydb data/students.bin
Welcome to the Tiny Database!
Loading the database...
Done!
> select fname=Yannick
Running query 'select fname=Yannick'
logs/1600954212483482624-select.txt
...
```

Ainsi, le programme pourra dès lors être lancé avec la redirection de stdin vers un fichier comme illustré ci-dessous. Ce fichier contient une requête par ligne.

```
$./tinydb data/students.bin < queries.txt
Welcome to the Tiny Database!
Loading the database...
```

```
Done!
Running query 'select fname=Cedric'
logs/1600954212483482724-select.txt
Running query 'update id=88 set lname=Quevrin'
logs/1600954212483482820-select.txt
Running query 'select fname=Damien'
logs/1600954212483482937-select.txt
...
```

2.5 Résultat des requêtes

Chaque requête renvoie un objet de type `query_result_t` illustré dans le Tableau 2.

query_result_t		
Champ	Type	Signification
students	`student_t*`	Pointeur vers la liste des étudiants concernés par la requête
size	`size_t`	Le nombre d'étudiants
query	`char[256]`	La requête à laquelle ce résultat se rapporte
start_ns	`long`	Le timestamp (en nanosecondes) du début de la requête
end_ns	`long`	Le timestamp (en nanosecondes) de la fin de la requête

TABLE 2 – La structure `query_result_t`.

Chacun de ces résultats doit être écrit dans un fichier à l'aide de la fonction `query_result_log`.

2.6 Organisation en processus

La base de données devra être implémentée de sorte que chaque type de requête (sauf `transaction`) soit gérée par un **processus dédié** : toutes les requêtes `select` seront gérées par le processus A, les requêtes `update` par le processus B, etc.

Le premier processus issu d'un lancement de l'exécutable tinydb est appelé, pour la suite de cet énoncé, le **processus principal**. Son rôle est de :

- créer les 4 processus dédiés (un par type de requête) ;
- récupérer les requêtes de l'utilisateur sur l'entrée standard (stdin) ;
- communiquer, à l'aide de pipes anonymes, les requêtes de l'utilisateur à ses enfants (les processus dédiés) ;
- traiter correctement les signaux (SIGINT et SIGUSR1) qui lui sont envoyés (voir Section 2.7 et Section 2.8 respectivement).

Lorsque le programme se termine (Section 2.7), le processus parent ferme tous les pipes et attend la fin des processus enfants avant de se terminer.

L'organisation des processus est illustrée dans la Figure 1 et les moyens de communication à implémenter au **minimum** (il est possible que plus soient nécessaires, ceci est laissé à votre appréciation) entre les processus sont illustrés dans la Figure 2.

2.7 Fin de la base de données

Le programme de base de données peut se terminer de plusieurs manières. Dans tous les cas, le programme se termine en sauvegardant le contenu de la base de données sur le disque à l'aide de la fonction `db_save` qui vous est fournie.

- Le fichier `stdin` arrive à sa fin (`CTRL+D`, ou fin du fichier redirigé)

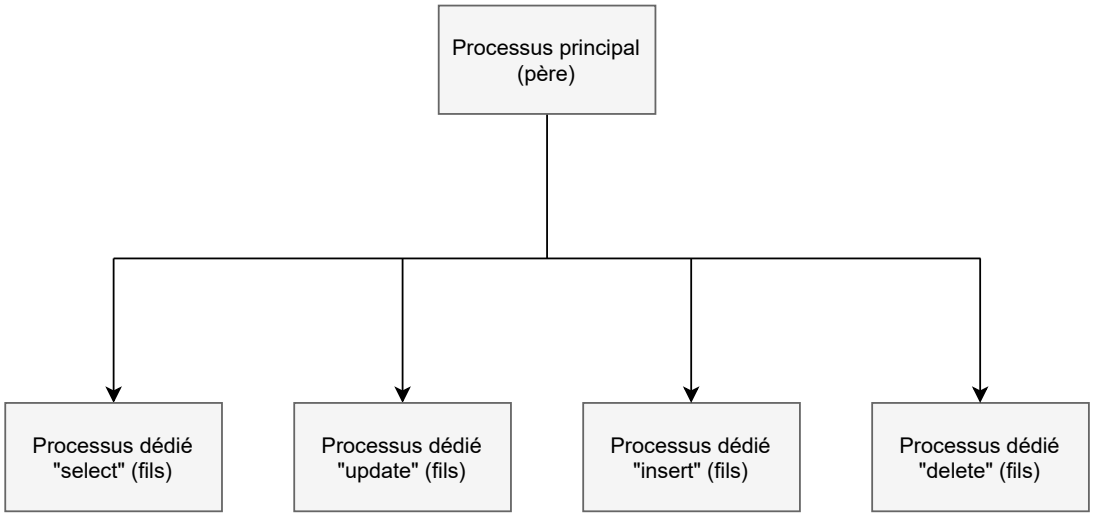

FIGURE 1 – Organisation en processus.

— Le processus principal capture un signal `SIGINT` (envoyé par exemple avec `CTRL+C`)

Voici un exemple de fin du programme de base de données via `CTRL+C`.

```
> select fname=Yannick
logs/1600954212483482624-select.txt
> ^C
Waiting for requests to terminate...
Committing database changes to the disk...
Done.
```

2.8 Sauvegarde sur le disque

Le processus principal de la base de données doit intercepter les signaux `SIGUSR1` qui lui sont envoyés et sauvegarder le contenu de la base de données sur le disque à l'aide de la fonction `db_save` qui vous est fournie le cas échéant. Les autres processus doivent ignorer ce signal.

3 Monitoring

Afin de surveiller le bon fonctionnement de votre programme de gestion de base de données, d'en surveiller l'état et d'en faciliter l'utilisation, vous devez écrire un script bash dont les fonctionnalités sont décrites dans cette section.

Ce script accepte au minimum un paramètre qui indique la fonctionnalité à exécuter.

Vous pouvez considérer que ce script sera toujours dans le même dossier que l'exécutable `tinydb`.

3.1 Lancement de la DB

La commande `run` permet de lancer le programme de base de données avec deux paramètres optionnels comme indiqué ci-dessous. Cette commande lance l'exécutable `tinydb` et affiche le PID du processus principal qui en découle.

```
$ ./monitoring run [<chemin_vers_la_db>] [-f <fichier_requetes>]
```

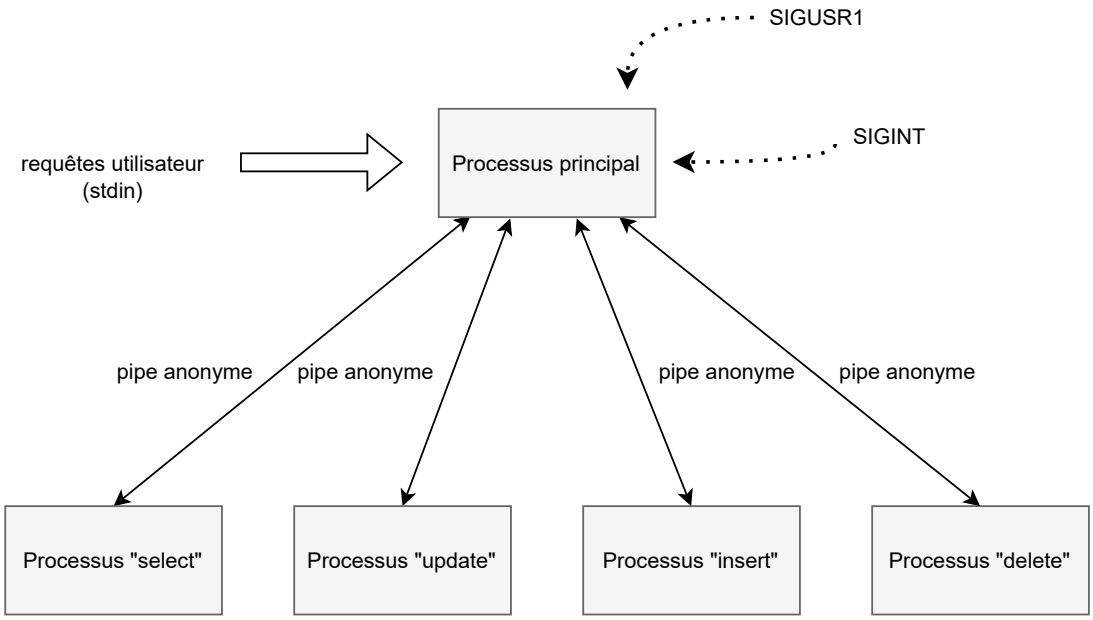

FIGURE 2 – Communications inter-processus (pipes anonymes et signaux) et via stdin.

Le chemin vers le fichier de base de données est optionnel et est mis à la valeur `data/students.bin` si aucune valeur n'est donnée. Si le fichier donné en paramètre n'existe pas, le script le crée (vide) en affichant un message d'avertissement sur `stderr`.

Le paramètre `-f` est aussi optionnel et donne le chemin vers un fichier contenant des requêtes (une par ligne, cf Section 2.4). Si ce fichier n'existe pas, le script s'arrête en affichant un message d'erreur.

3.2 Vérification de l'état de la base de données

La commande `status` donne des informations sur l'état des bases de données en cours d'exécution (au minimum les PID de leur processus principal) et s'utilise comme indiqué ci-dessous.

```
$ ./monitoring status
1 instance running
      > PID 1990
```

3.3 Sauvegarde sur le disque

La commande `sync` envoie un signal `SIGUSR1` au processus principal de chaque base de données en cours d'exécution pour leur demander que leur base de données soit sauvegardée sur le disque.

```
$ ./monitoring sync
Found 2 instances of tinydb
  Sync process 1990...
  Sync process 2546...
Done
```

3.4 Arrêt de la base de données

La commande `shutdown` permet d'envoyer un signal qui arrête la base de données comme indiqué ci-dessous.

```
$ ./monitoring shutdown [<pid>]
```

Cette commande prend un paramètre optionnel `pid` qui est le PID du processus principal à arrêter. Si ce PID n'est pas celui d'un processus principal, le script affiche un message d'erreur sur stderr. Si aucun PID n'est donné, le script demande confirmation pour chaque processus principal qui tourne.

4 Spécifications techniques

4.1 Synchronisation

Comme la base de données est partagée entre plusieurs processus il faut que cette base de données soit explicitement partageable entre processus. Utilisez `mmap` ou `shmget` afin de gérer la base de données avec de la mémoire partagée. Ce partage de mémoire peut introduire des problèmes de concurrence mais vous ne devez pas les gérer dans le cadre de ce projet.

4.2 Transactions

Une transaction délimite un ensemble de requêtes qui doivent être exécutées avant d'exécuter celles de la transaction suivante. On commence une transaction avec le mot-clef `transaction` et celle-ci se termine lors du prochain mot-clef `transaction` rencontré ou lorsque le programme se termine.

Voici un exemple avec les transactions, les requêtes et les résultats des requêtes en commentaire (précédé d'un #).

```
transaction
select id=88
# retourne Yannick Molinghen in section INFO born on the 14/02/1995
transaction
update id=88 set fname=Alexis
select id=88
# Ces deux requêtes peuvent être exécutées en parallèle.
# Le résultat peut donc donner un étudiant dont le fname est Yannick, Alexis
# ou un mélange des deux, comme Alenick
transaction
select id=88
# Retourne Alexis Molinghen in section INFO born on the 14/02/1995
```

Ainsi, l'utilisateur peut se prémunir des modifications concurrentes en englobant chaque requête qui modifie la base de données dans une transaction individuelle.

5 Ce qui est mis à votre disposition

Vous pouvez télécharger la base du projet sur l'Université Virtuelle afin de ne pas commencer de zéro (ce n'est pas obligatoire). Ce répertoire contient :

- quelques headers et fichiers C pour ne pas partir de zéro,
- une base de données de plusieurs milliers d'étudiants dans `data/students.bin`,
- des tests automatiques lançables avec la commande `make tests -B`,
- un makefile à compléter.

6 Critères d'évaluation

Votre projet doit compiler avec g++ version 9.4 (ou ultérieur) et les flags ci-dessous (présents dans le makefile fourni). Si votre programme ne compile pas, vous recevrez une note de 0/20.

```
FLAGS=-std=c++17 -Wall -Werror -Wpedantic -D_GNU_SOURCE
```

6.1 Pondération

- Tests automatiques **/3**
- Ce projet de programmation système va principalement évaluer votre compétence à manier correctement les outils liés aux systèmes d'exploitation (threads, fichiers, sémaphores, ...) dans le langage C. La pertinence des outils utilisés ainsi que la manière dont ils sont utilisés (trop, pas assez, au mauvais endroit, trop longtemps, ...) sont evalués. **/5**
- Bash **/5**
- Ce projet doit contenir un rapport dont la longueur attendue est de deux à trois pages (max 5). Ce rapport décrira succinctement le projet, les choix d'implémentation si nécessaire, les difficultés rencontrées et les solutions originales que vous avez fournies. **/5**
  - Orthographe
  - Structure
  - Légende des figures
- Votre code sera aussi évidemment examiné en termes de clarté, de documentation, de commentaires et de structure. **/2**

7 Remise du projet

Vous devez remettre un projet par groupe contenant un fichier zip contenant

- vos sources
- un Makefile
- votre rapport au format PDF
- les tests que vous auriez écrits

N'incluez ni le fichier de base de données de milliers d'étudiants ni vos logs dans le fichier zip !

Vous devez soumettre votre projet sur l'**Université Virtuelle** pour le **7 novembre 2022 23h59** au plus tard.

Retards

Tout retard sera sanctionné d'un point par tranche de 4h de retard, avec un maximum de 24h de retard et devra être soumis sur l'université virtuelle.

Questions

Le meilleur moment pour poser vos questions sera pendant les séances de travaux pratiques. Vous pouvez cependant aussi poser vos questions via email à yannick.molinghen@ulb.be si nécessaire.