Portail IE
Module INF2
Principes de systèmes informatiques

TP Système de gestion de fichiers

25 février 2021

On se propose de reproduire de façon très schématique les fonctions d'un système de fichiers sur disque en représentant un disque comme un fichier. Les principales hypothèses simplificatrices sont que tout contenu de fichier tient sur un secteur de disque, ainsi que toute table représentant un répertoire. Même schématique, le système obtenu permet de mettre en évidence l'allocation des secteurs du disque, le rôle du cache d'accès au disque (le cache de données) et le rôle du cache d'accès aux répertoires (le cache de répertoires).

1 Spécification

Un disque est simulé en utilisant un fichier de l'environnement de l'étudiant. Le nom du fichier est le nom qui aura été passé à l'opération de création du disque. La structure de disque en liste de secteurs est simulée sur le fichier en y écrivant une liste d'autant d'éléments que de secteurs. Le numéro d'un secteur n'est pas écrit dans le secteur ; le rang d'un secteur dans la liste indique son numéro. Un secteur est toujours représenté, qu'il soit vide ou non[1]. On suppose qu'un secteur peut contenir n'importe quel fichier ou répertoire. Les secteurs sont donc extensibles (contrairement aux vrais secteurs d'un disque dur).

Un mode trace permet de montrer tous les accès au disque. Il peut être activé en exécutant `fs_log_on()`, et désactivé en exécutant `fs_log_off()`.

- L'accès au disque se fait au travers d'un cache de données qui a pour but d'amortir le coût de l'accès au disque en exploitant le principe de localité, et en désynchronisant les écritures sur le disque. Le contenu du cache est représenté par une liste de copies de secteurs. Pour chaque secteur copié, un bit *dirty* indique si la copie a subi une modification qui n'a pas été propagée à l'original, et un bit *used* indique si la copie a été utilisée récemment. Ces bits servent à organiser la resynchronisation du cache de données et du disque et à organiser le remplacement des copies de secteurs les moins utilisés (voir le cours).
- Le système de fichiers ne lit les données contenues dans le disque que via le cache de données. Au cas où la donnée recherchée n'est pas déjà dans le cache, le système de fichiers effectue d'abord une copie de la donnée dans le cache, pour ensuite seulement y lire les données. C'est aussi vrai pour les écritures. Une donnée est toujours modifiée via sa copie dans le cache de données en y inscrivant sa nouvelle valeur. La nouvelle valeur sera propagée de la copie à l'original de la donnée à un autre moment, éventuellement après que d'autres modifications aient été apportées dans la copie. Ainsi, plusieurs écritures via le système de fichiers peuvent ne provoquer qu'une écriture sur le disque. Si la copie n'est pas présente dans le cache, le système de fichiers doit commencer par lire sur le disque le secteur originel. Ainsi, une écriture via le système de fichiers peut d'abord causer une lecture du disque. Le mode trace montre les accès au cache de données et indique si la donnée a été trouvée dans le cache (*cache hit* en anglais) ou non (*cache miss* en anglais).
- L'accès aux répertoires et aux fichiers se fait au travers d'un cache de répertoires (le dcache pour *directory cache*) qui représente en mémoire une partie de la hiérarchie des répertoires. À un chemin d'accès qui mène à un répertoire, le cache de répertoires fait correspondre une liste de descripteurs des éléments de ce répertoire (un dictionnaire). On trouve dans ces descripteurs le nom, le type (fichier ou répertoire), et le secteur occupé par cet élément. Tous les répertoires n'ont pas vocation à être représentés dans le cache de répertoire, mais à tout moment, un répertoire reçoit une attention particulière. Il s'agit du répertoire de travail. Comme tous les accès via des noms de fichiers ou de répertoires se font par rapport au réper-

[1] En fait, un secteur n'est jamais vide, il est tout au plus pas utilisé ou pas initialisé.

toire de travail, le cache de répertoire doit toujours contenir la description du répertoire de travail et celle de ces ascendants. Le cache de répertoires est structuré en nœuds qui correspondent chacun soit à un fichier, soit à un répertoire. Un nœud mémorise l'association entre un nom d'objet et le secteur disque sur lequel il est représenté. Du point de vue du programmeur, un nœud de type répertoire est construit à l'aide du constructeur `DNODE` et un nœud de type fichier par un constructeur `FNODE`. Par exemple, `FNODE(`*n*`,`*s*`)` crée un nœud de type fichier qui associe le secteur *s* au nom *n*. Les accesseurs `is_DNODE` et `is_FNODE` permettent de déterminer si un nœud est celui d'un répertoire ou d'un fichier. Enfin, quel que soit le type du nœud, l'accesseur `name_of_node` permet d'obtenir le nom qu'il décrit et l'accesseur `nb_of_node` permet d'obtenir le numéro de secteur associé. Par exemple,

`is_DNODE(FNODE(`*n*`,`*s*`))`

vaut `False` et

`name_of_node(FNODE(`*n*`,`*s*`))`

vaut *n*.

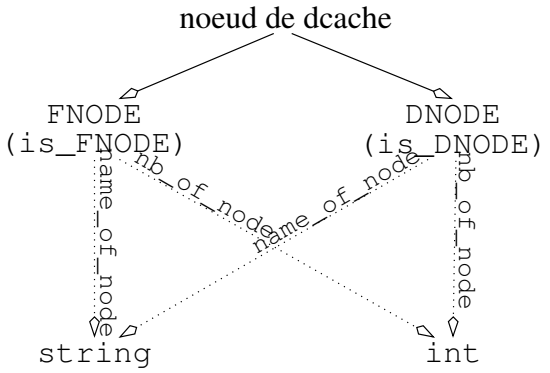

Figure 1 – La manipulation des nœuds du cache

2 La bibliothèque fs.py

Une bibliothèque de fonctions Python réalise les couches basses du système de fichiers. Les principales fonctions sont les suivantes :

- `fs_make_disk(`*n*`,`*s*`)` : crée un disque de nom *n* et de taille *s*. Le nom *n* est celui que porte le fichier qui simule le disque. La taille *s* est mesurée en nombre de secteurs.
- `fs_init_cache(`*s*`)` : initialise un cache de données de taille *s*. La taille s est mesurée en nombre de secteurs.
- `fs_format_disk(`*n*`,`*s*`)` : formate le disque de nom *n* et de taille *s*. Cela nécessite quelques écritures sur le disque, qui sont faites via le cache de données, qui doit donc avoir été initialisé au préalable.
- `fs_init_dcache()` : initialise le cache de répertoires (ou dcache).
- `fs_dump_disk(`*n*`)` : affiche le contenu du disque de nom *n*.
- `fs_dump_cache()` : affiche le contenu du cache de données.
- `fs_dump_dcache()` : affiche le contenu du cache de répertoires.
- `fs_sync_cache()` : synchronise le cache de données et le disque.
- `fs_list_in_dcache()` : retourne le contenu du répertoire de travail courant sous la forme d'une liste de nœuds du cache de répertoires.
- `fs_lookup_in_dcache(`*n*`)` : retourne `False` si aucun objet du répertoire de travail courant ne porte le nom *n*. Retourne le nœud correspondant au nom n sinon.
- `fs_cd_in_dcache(`*n*`)` : affecte le nom *n* au répertoire de travail.
- `fs_getfreeblock()` : demande un secteur de disque. Retourne `False` si aucun secteur est disponible ; un numéro de secteur sinon. Dans ce dernier cas, le système enregistre que ce secteur est alloué.
- `fs_mkdir_in_dcache(`*n*`,`*s*`)` : crée un nouveau répertoire de nom *n* dans le répertoire de travail courant et le représente dans le secteur *s*.
- `fs_touch_in_dcache(`*n*`,`*s*`)` : crée un nouveau fichier de nom *n* dans le répertoire de travail courant et le représente dans le secteur *s*.
- `fs_rm_in_dcache(`*n*`)` : supprime l'objet de nom *n* du répertoire de travail courant.
- `fs_write_in_dcache(`*n*`,`*s*`)` : affecte au fichier de nom *n* le contenu *s*.
- `fs_read_in_dcache(`*n*`)` : retourne le contenu du fichier de nom *n*.
- `fs_cwd()` : retourne le nom du répertoire courant.
- `fs_stats()` : affiche les statistiques hit/miss du cache.
- `name_of_node(`*n*`)` : retourne le nom du nœud *n*.
- `nb_of_node(`*n*`)` : retourne le numéro de secteur du nœud *n*.
- `is_DNODE(`*n*`)` : indique si *n* est un nœud du cache qui représente un secteur qui contient un répertoire. Retourne `True` ou `False`.
- `is_FNODE(`*n*`)` : indique si *n* est un nœud du cache qui représente un secteur qui contient un fichier. Retourne `True` ou `False`.
- Les variables `FS_DISK`, `FS_CACHE`, `FS_CACHE_SIZE`, `FS_DCACHE` et `FS_CURRENT_DIR` contiennent respectivement le nom du disque, le cache de données, sa taille, le cache de répertoires et le répertoire de travail courant. Le contenu de ces variables est positionné par les fonctions de la bibliothèque `fs.py`. Il ne faut pas essayer de les positionner directement. Elles servent à désigner directement certains composants du système de fichiers. Elles peuvent être utilisées dans les programmes qui utilisent les fonctions de la bibliothèque `fs.py`. Leur fonctionnement est le suivant :
  - `FS_DISK` est positionnée par `fs_make_disk` ;
  - `FS_CACHE` et `FS_CACHE_SIZE` sont positionnées par `fs_init_cache` et `FS_CACHE` est mise à jour par `fs_sync_cache`, `fs_read_in_cache` et `fs_write_in_cache` ;
  - `FS_DCACHE` est positionnée par `fs_init_dcache` et mise à jour par `fs_touch_in_dcache`, `fs_mkdir_in_dcache`, `fs_cd_in_dcache` et `fs_rm_in_dcache` ;
  - enfin, `FS_CURRENT_DIR` est positionnée par `fs_init_dcache` et mise à jour par `fs_cd_in_dcache` (voir figure 2).

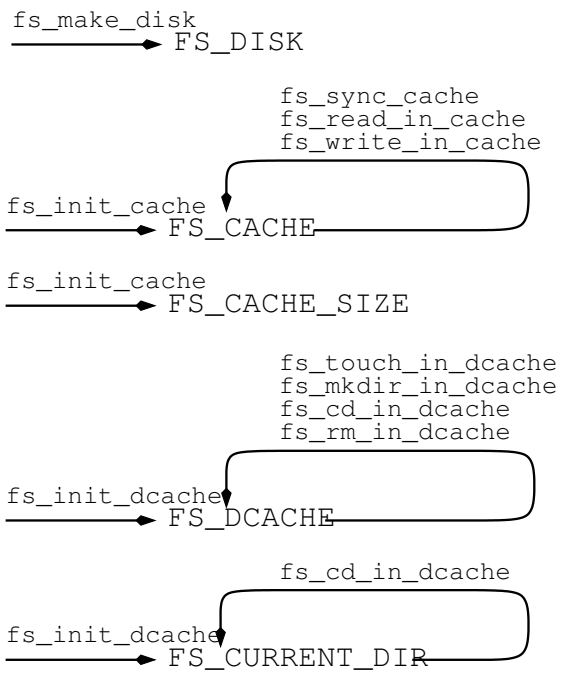

Figure 2 – Les variables globales de `fs.py`

3 Plan de développement

Le plan de développement propose un ordonnancement des étapes du développement d'un logiciel. Dans le cas présent, le logiciel consiste en un unique fichier, `tp_sgf.py`, dans lequel les fonctions à réaliser restent à compléter. Certaines sont déjà complétées, et il conviendra alors de consulter et analyser la façon dont elles ont été programmées. Cette façon de faire pourra resservir pour d'autres fonctions.

Fonction INIT : Programmer la fonction INIT qui initialise et formate le disque, le cache de données et le cache de répertoires. Les trois paramètres de la fonction INIT sont le nom du disque (en fait, le nom du fichier qui joue le rôle du disque), la taille du disque, et la taille du cache de données.

```
;; Initialize disk, cache and dcache.
;; INIT: string * int * int * int
                                -> proc
```

Fonction DUMP : Programmer la fonction DUMP qui affiche (`print`) les contenus du disque, du cache de données et du cache de répertoires. La fonction DUMP permet de tester les commandes du système de fichiers. En effet, certaines d'entre-elles ne retournent rien, mais changent les contenus du disque, du cache de données ou du cache de répertoire. On pourra donc utiliser la fonction DUMP pour valider ces commandes.

```
;; Dump disk, cache and dcache.
;; DUMP: proc
```

Utiliser la fonction DUMP pour tester la fonction INIT. Qu'observe-t-on ? Justifier.

Fonction LS : Programmer la fonction LS qui retourne le contenu du répertoire de travail courant, ou qui teste si à un nom passé en paramètre correspond un objet du répertoire de travail courant[2]. Dans le premier cas, le résultat est la liste des noms des objets du répertoire courant. Dans le second cas, le résultat est soit le nom passé en paramètre, si un objet de ce nom existe, soit `False` .

```
;; List contents of current working
;; directory.
;; LS : [name]
                -> ((list name)|name|False)
```

Fonction SYNC : Programmer la fonction SYNC qui resynchronise le cache de données et le disque[3].

```
;; Flush pending write operations from
;; the data cache to the disk.
;; SYNC : proc
```

Utiliser la fonction SYNC juste après INIT, puis faire DUMP. Comparer avec l'expérience précédente. Examiner les bits *dirty* et *used*.

Fonction MKDIR : Programmer la fonction MKDIR qui crée un nouveau répertoire dans le répertoire de travail courant[4]. Le nom du nouveau répertoire est passé en paramètre. La fonction retourne `True` si tout se passe bien, `False` cas d'anomalie, et alors affiche un message.

```
;; Create a new directory in the current
working directory.
;; MKDIR: string -> bool
```

Cette fonction n'a le comportement annoncé que si aucun objet déjà existant dans ce répertoire ne porte le nom passé en paramètre et s'il y a assez de place sur le disque pour représenter le nouveau répertoire. Dans un premier temps, ne pas vérifier que le nom ne désigne rien, ni qu'il y a assez de place ; le faire dans un second temps (ex. quand toutes les commandes auront été programmées).

Tester la fonction MKDIR en utilisant les fonctions DUMP et SYNC.

Fonction CD : Programmer la fonction CD qui change le répertoire de travail[5]. Cette fonction prend un paramètre qui peut être n'importe quel nom de répertoire connu dans le répertoire de travail courant, y compris "`.`" et "`..`". La fonction retourne `True` si tout se passe bien, et affiche un message d'erreur en cas d'anomalie.

```
;; Change working directory.
;; CD: string -> bool
```

Dans un premier temps, ne pas vérifier que le nom désigne un répertoire dans le répertoire de travail courant ; le faire dans un second temps.

[2] Faire `man ls` dans une fenêtre de commande UNIX pour se rendre compte de toutes les possibilités de la vraie commande ls.

[3] Faire `man sync`.

[4] Faire `man mkdir`.

[5] Faire `man cd` pour une vue plus complète de la commande cd.

Fonction TOUCH : Programmer la fonction TOUCH qui crée un nouveau fichier dans le répertoire de travail courant[6]. Le nom du nouveau fichier est passé en paramètre. La fonction retourne `True` si tout se passe bien, `False` cas d'anomalie, et alors affiche un message.

```
;; Create a new file in the current
;; working directory.
;; TOUCH: string -> bool
```

Dans un premier temps, ne pas vérifier que le nom ne désigne rien, ni qu'il y a assez de place ; le faire dans un second temps.

Tester la fonction TOUCH en utilisant les fonctions DUMP et SYNC.

Fonction WRITE : Programmer la fonction WRITE qui écrit une chaîne de caractères dans un fichier. Le premier paramètre est le nom du fichier où écrire, et le second paramètre est la chaîne de caractère. La fonction retourne `True` si tout se passe bien, `False` cas d'anomalie, et alors affiche un message.

```
;; Write a string in a file.
;; WRITE : string * string -> bool
```

Dans un premier temps, ne pas vérifier que le nom désigne bien un fichier ; le faire dans un second temps.

Tester la fonction WRITE en utilisant les fonctions DUMP et SYNC.

Fonction READ : Programmer la fonction READ qui lit le contenu d'un fichier. Le paramètre est le nom du fichier à lire. La fonction retourne le contenu lu si tout se passe bien, `False` cas d'anomalie, et alors affiche un message.

```
;; Read in a file.
;; READ: string -> (string|False)
```

Dans un premier temps, ne pas vérifier que le fichier existe dans le répertoire de travail courant ; le faire dans un second temps.

Fonction RM : Programmer la fonction RM qui supprime un élément d'un répertoire[7]. Le nom de l'élément à supprimer est passé en paramètre. La fonction retourne `True` si tout se passe bien, `False` cas d'anomalie, et alors affiche un message.

```
;; Remove an element of a directory.
;; RM: string -> bool
```

Cette fonction n'a le comportement annoncé que si le nom passé en paramètre désigne quelque chose dans le répertoire de travail courant. De plus, si l'élément désigné est un répertoire, celui-ci doit être vide. Dans un premier temps, ne pas vérifier que le nom désigne quelque chose, ni le cas échéant que le répertoire est vide ; le faire dans un second temps.

Tester la fonction RM en utilisant les fonctions DUMP et SYNC.

4 Expérimentations

Concevoir des scénarios qui mettent en évidence les effets du cache et ses conditions de bon fonctionnement.

Pour cela, partir d'un scénario de création/exploration de répertoires et de fichiers qui occupe une centaine de secteurs disque. Ensuite, faire varier ce scénario afin de mettre en évidence les rôles des différents paramètres, taille du cache et localité des accès, ainsi que les effets du cache, éviter des accès disques et la désynchronisation des opérations du disque par rapport aux actions sur le système de fichiers.

- La mesure de l'efficacité du cache se fait par le *hit ratio* (*hit/(hit+miss)* où *hit* représente le nombre d'accès disque où le secteur a été trouvé dans le cache, et *miss* représente le nombre d'accès disque où le secteur n'a pas été trouvé dans le cache). Celui-ci est fourni par le simulateur de disque quand il fonctionne en mode trace.
- Taille de cache : jouer ce scénario et ses variantes avec différentes valeurs de la taille du cache de données. Qu'observe-t-on ?
- Localité des accès : concevoir des scénarios d'accès au disque qui respectent plus ou moins la localité des accès. Par exemple, un scénario d'édition de très peu de fichier dans très peu de répertoires, contre un scénario d'édition de beaucoup de fichiers dans beaucoup de répertoires.
- Désynchronisation : dans les scénarios précédents, repérer les opérations qui sont censées lire le disque (CD, LS et READ) et les opérations sont censées l'écrire (MKDIR, TOUCH et WRITE), et vérifier dans la trace si les accès au disque correspondent. Ajouter des appels à l'opération SYNC pour voir.

Optionnel : Résumer les observations dans une courbe 3D :

taille du cache × localité → hit ratio.

Utiliser pour cela la librairie (`https://cpge.frama.io/fiches-cpge/Python/Graphiques/5-3D/`).

[6] Faire `man touch` pour voir les autres fonctionnalités de cette commande.

[7] Faire `man rm` pour en savoir plus.