Annexe E

Le nommage

E.1 Le système de fichiers

On se propose de reproduire de façon très schématique les fonctions d'un système de fichiers 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 *dcache*).

E.1.1 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 Scheme 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.
  Un mode trace permet de montrer tous les accès au disque. Il peut être activé en exécutant `(define fs-log #t)`, et désactivé en exécutant `(define fs-log #f)`.
- 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.
  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 de donnée, 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 de données, 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 (ou *dcache*) qui représente en mémoire une partie de la hiérarchie des répertoires. À un chemin d'accès qui mène à un répertoire, le dcache fait correspondre une liste de descripteurs des éléments de ce répertoire. 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 dcache, 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 répertoire de travail, le dcache 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 `DNODE?` et `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 `NODE->name` permet d'obtenir le nom qu'il décrit et l'accesseur `NODE->nb` permet d'obtenir le numéro de secteur as-

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

socié. Par exemple, `(DNODE? (FNODE` n s`))` vaut `#f` et `(NODE->name (FNODE` n s`))` vaut n.

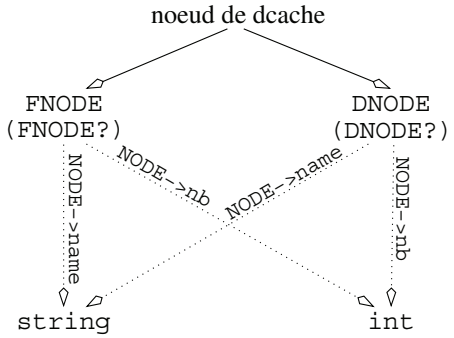

FIGURE E.1 – Les constructeurs et accesseurs des nœuds de dcache

E.1.2 La bibliothèque `fs.ss`

Une bibliothèque de fonctions Scheme 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 blocs.
- `(fs-init-cache` s`)` : initialise un cache de données de taille s. La taille s est mesurée en nombre de blocs.
- `(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.
- `(fs-init-dcache)` : initialise le cache de répertoire (ou dcache pour *directory cache*).
- `(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 dcache.
- `(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 dcache.
- `(fs-lookup-in-dcache` n`)` : retourne `#f` 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 `#f` si aucun secteur est disponible ; un numéro de secteur sinon. Dans ce dernier cas, le système enregistre que ce bloc 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.
- Les variables `FS-DISK`, `FS-CACHE`, `FS-CACHE-SIZE`, `FS-DCACHE` et `FS-CURRENT-DIR` contiennent respectivement le nom du disque, le cache, sa taille, le dcache et le répertoire de travail courant. Le contenu de ces variables est positionné par les fonctions de la bibliothèque `fs.ss`. Il ne faut pas essayer de les positionner directement. Elle 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.ss`.
  `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 E.2).

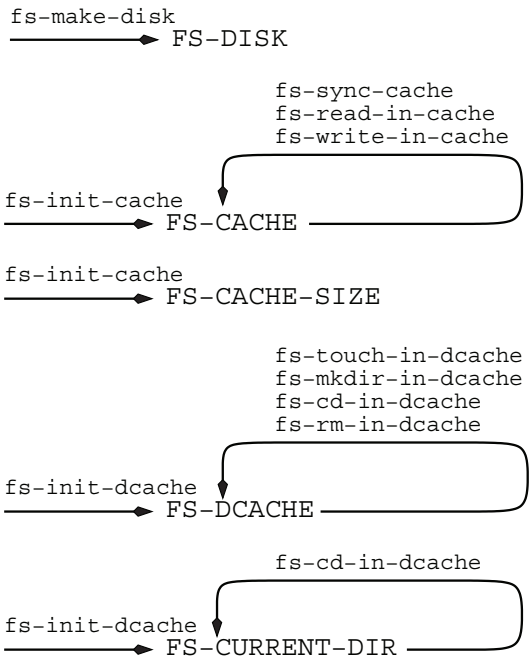

FIGURE E.2 – Les variables globales de `fs.ss`

E.1.3 Plan de développement

1. Programmer la fonction `INIT` qui initialise et formate le disque, le cache de données et le dcache. Les quatre 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, la taille du cache de données et la taille du dcache.

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

Programmer la fonction DUMP qui affiche les contenus du disque, du cache et du dcache. 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 ou du dcache. 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.

2. Programmer la fonction LS qui affiche 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 #f (faux en Scheme).

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

3. 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*.

4. 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.

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

Cette fonction n'a le comportement annoncé que si aucun objet déjà existant dans ce répertoire ne porte le nom passé en paramètre et si 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).

5. 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 "..".

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

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.

6. 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.

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

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.

7. Programmer la fonction WRITE qui écrit une chaîne de caractère dans un fichier. Le premier paramètre est le nom du fichier où écrire, et le second paramètre est la chaîne de caractère.

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

Programmer la fonction READ qui lit le contenu d'un fichier. Le paramètre est le nom du fichier à lire.

```
;; Read in a file.
;; READ: string -> string
```

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

8. Programmer la fonction RM qui supprime un élément d'un répertoire [7]. Le nom de l'élément à supprimer est passé en paramètre.

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

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.

9. Concevoir un scénario de création/exploration de répertoires et de fichiers qui occupe une centaine de secteurs disque. Jouer ce scénario avec différentes valeurs de la taille du cache de données. Qu'observe-t-on ?

   Concevoir un scénario qui mette en évidence le fait que les écritures sur disque ne se font pas forcément dans l'ordre où l'utilisateur naïf peut penser qu'elles sont faites. C'est-à-dire qu'à une écriture via le système de fichiers peut correspondre une lecture disque, et qu'à plusieurs écritures via le système de fichiers il peut ne correspondre qu'une écriture disque.

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`.
6. Faire `man touch` pour voir les autres fonctionnalités de cette commande.
7. Faire `man rm` pour en savoir plus.

E.2 Le port USB

Connecter la plus grande variété possible de périphériques USB sur votre système hôte préféré, et examiner comment ce dernier voit les périphériques.

Par exemple sur MacOS, la carte du système donne ceci pour les ports USB externes.

```
Bus USB à grande vitesse :

  Emplacement du contrôleur de l'hôte :
USB intégré
  Gestionnaire du contrôleur de l'hôte :
AppleUSBEHCI
  Identifiant du périphérique PCI : 0x27cc
  Identifiant de révision du PCI : 0x0002
  Identifiant du fournisseur du PCI : 0x8086
  Numéro bus : 0xfd

iPod :

  Capacité : 7.41 Go
  Support amovible : Oui
  Disque amovible : Oui
  Nom BSD : disk2
  Version : 0.01
  Alimentation du BUS (mA) : 500
  Vitesse : Jusqu?à 480 Mb/s
  Fabricant : Apple Inc.
  Gestionnaires OS9 : Non
  Identifiant du produit : 0x1262
  Numéro de série : 000A27001AF15C1F
  État S.M.A.R.T. : Non géré
  Identifiant du fournisseur :
0x05ac  (Apple Computer, Inc.)
  Volumes :
iPod (Jean Durand) :
  Capacité : 7.41 Go
  Disponible : 3.09 Go
  Inscriptible : Oui
  Système de fichiers : Journaled HFS+
  Nom BSD : disk2s2
  Point de montage : /Volumes/iPod (Jean Durand)

USB DISK 2.0 :

  Capacité : 3.77 Go
  Support amovible : Oui
  Disque amovible : Oui
  Nom BSD : disk1
  Version : 1.00
  Alimentation du BUS (mA) : 500
  Vitesse : Jusqu'à 480 Mb/s
  Fabricant :
  Gestionnaires OS9 : Non
  Identifiant du produit : 0x3100
  Numéro de série : 07870C3E5415
  État S.M.A.R.T. : Non géré
  Identifiant du fournisseur : 0x13fe

iSight intégrée :
  Version : 1.84
  Alimentation du BUS (mA) : 500
  Vitesse : Jusqu'à 480 Mb/s
  Fabricant : Micron
  Identifiant du produit : 0x8501
  Identifiant du fournisseur :
0x05ac  (Apple Computer, Inc.)
```

On peut y voir le niveau d'alimentation du port USB, 500 mA, la vitesse maximale supportée, 480 Mb/s, des identifiants de produit, de série et de fournisseur qui viennent du périphérique lui-même, et des indications sur la fonction du périphérique : disque amovible, capacité, etc. Concernant la capacité, les valeurs bizarres comme 3,77 Go et 7,41 Go sont tout simplement l'expression en unité informaticienne (c-à-d. puissances successives de 2^{10}) des capacités indiquées par le fournisseur en unité SI (c-à-d. puissances successives de 10^3). Sur l'emballage, ces capacités étaient de 4 Go et 8 Go. Les écarts sont quand même de près de 300 Mo et 600 Mo.