Résumé du chapitre 5
Systèmes de gestion des fichiers

Jacques Mossière

22 novembre 2005

1 Introduction

Le système de gestion des fichiers (SGF) est la partie du système d'exploitation chargée du stockage et de la récupération d'informations. Plus précisément, les systèmes de fichier ont trois objectifs principaux.

- Permettre le stockage **permanent** d'informations. Nous entendons par permanence non seulement que les informations doivent survivre à la fin d'exécution d'un processus ou à la mise hors tension de la machine, mais qu'elles doivent pouvoir être conservées de façon fiable pendant des décennies, c'est à dire pendant un temps supérieur à la vie d'un ordinateur ou d'un disque compact.
- Permettre la **mise en commun d'informations** entre utilisateurs, que l'utilisation se fasse de façon simultanée ou non ; cette mise en commun peut également impliquer un ensemble de machines connectées par réseau.
- Permettre le stockage d'**unités d'informations de grande taille**, supérieure à la taille de la mémoire centrale.

L'unité de conservation d'informations est appelée un **fichier**. Un utilisateur peut désigner un fichier par un système de désignation symbolique. Comme tout objet dans un système, un fichier peut être manipulé par un langage de commande ou par des appels système.

Dans ce chapitre, nous passons en revue brièvement l'organisation logique des fichiers (§ 2), puis la désignation et les catalogues (§ 3). Nous montrons ensuite comment réaliser un système de gestion des fichiers (§ 4) et enfin nous décrivons quelques méthodes pour assurer la fiabilité du SGF (§ 5) tant du point de vue de la résistance aux pannes que du point de vue de la résistance aux malveillances. La réalisation du SGF est illustrée d'exemples ponctuels tirés d'Unix. Nous donnons enfin (§ 6) une description complète du système de gestion des fichiers de PedagOS, des principes de base de sa réalisation et des structures de données sur disque et en mémoire principale, ainsi qu'une part significative des programmes du SGF.

2 Organisation logique des fichiers

2.1 Généralités

Un fichier est utilisé pour conserver des informations de différentes sortes : document d'un programme de traitement de texte, programme source, programme objet, binaire exécutable, etc.

Pour éviter des manipulations saugrenues, il est classique de placer au début de chaque fichier un en-tête qui précise la nature des informations contenues dans le fichier et qui permet aussi de retrouver l'application qui doit en général accéder au fichier.

Exemple. Les fichiers Unix qui contiennent du JPEG ont l'en-tête 0xFFD8 et les fichiers ELF 0x7F45.

De même, pour faciliter l'identification par l'utilisateur du type des données conservées, il est classique de placer dans le nom symbolique d'un fichier un suffixe associé à ce type (`toto.c` sera ainsi interprété comme un fichier ascii contenant un programme C ; `toto.o` sera un binaire objet et `toto` un programme exécutable).

En fait, le SGF ne conserve pas seulement des informations sur le contenu d'un fichier, mais aussi d'autres informations comme la taille du fichier, sa date de création, etc. Ces informations sont appelées les **attributs** du fichier.

2.2 Structure

Un fichier peut être organisé comme une simple suite d'octets, auquel cas l'organisation interne d'un fichier ne relève que des applications qui l'utilisent, ou plus ou moins structuré. Deux modes de structuration ont été couramment utilisés.

- La structuration en une suite d'articles, l'article (« record ») étant l'unité logique d'accès au fichier. Cette structuration était calquée sur les périphériques anciens (imprimantes, lecteurs de cartes) qui imposaient des transferts par blocs de caractères de taille fixe (longueur d'une ligne ou d'une carte).
- La structuration dite en mode séquentiel indexé. L'idée était de repérer chaque article d'un fichier par un identificateur unique, ou clé, et de permettre soit de parcourir séquentiellement le fichier, dans l'ordre des clés, soit d'accéder directement à un article de clé donnée. Les fichiers séquentiels indexés étaient utilisés dans les applications de gestion avant la généralisation des systèmes de gestion de bases de données.

À l'heure actuelle, la tendance est à l'utilisation de fichiers non structurés qui ne font aucune hypothèse sur les structures dont ont besoin les applications et qui permettent de rediriger simplement les entrées et sorties standard.

2.3 Accès

Les SGF fournissent deux modes d'accès aux fichiers, le mode **séquentiel** dans lequel un processus peut lire ou écrire séquentiellement le contenu du fichier, à partir du début, et le mode **direct** ou **aléatoire** dans lequel les transferts peuvent avoir lieu à n'importe quelle position dans le fichier. Ce dernier mode n'est autorisé que si le support des fichiers est lui-même à accès aléatoire, ce qui est en général le cas (disques magnétiques).

Alors qu'on pourrait croire que le mode séquentiel est un simple héritage du passé et des dérouleurs de bande, ce n'est absolument pas le cas et le mode séquentiel, pour des raisons liées à la conception des algorithmes (« la pensée humaine est séquentielle »), est de très loin le mode d'accès le plus fréquemment utilisé.

2.4 Opérations

Il n'est pas question de passer en revue ici l'ensemble des appels système de gestion des fichiers, ni de fournir des exemples de leur utilisation. Chacun pourra se reporter pour cela au manuel de son système préféré. En dehors des opérations de transfert proprement dites, les SGF offrent des appels systèmes liés à la création, à la destruction et à la désignation des fichiers, ainsi que des appels systèmes d'ouverture et de fermeture. Le but de l'ouverture est de vérifier qu'un fichier existe et est bien accessible par un processus, ainsi que de récupérer, dans des tables en mémoire centrale, les attributs du fichier et la description de son implantation sur disque. Ces informations sont supprimées à la fermeture du fichier.

3 Désignation et catalogues

Pour un utilisateur humain, la méthode de désignation la plus agréable est d'utiliser des identificateurs symboliques. C'est le cas des SGF ou chaque fichier est désigné par un nom sous forme de chaîne de caractères. A partir d'un nom, le SGF doit être capable de retrouver les attributs du fichier et sa localisation sur disque. Il utilise pour ce faire un ensemble de tables de correspondances appelées **catalogues** ou répertoires.

Compte tenu du grand nombre de fichiers existants et de la multiplicité des utilisateurs qui veulent partager des fichiers, mais aussi désigner de façon autonome leurs fichiers privés, il est impératif de structurer l'espace des noms. Une structure commode est la structure d'arbre utilisée comme suit : aux feuilles de l'arbre se trouvent les fichiers, aux nœuds intermédiaires des catalogues. Un nom de fichier est composé d'une suite d'identificateurs, chacun des identificateurs étant un nom présent dans un catalogue intermédiaire. Le catalogue figurant à la racine de l'arbre est appelé catalogue racine.

Sur la figure 1, `/A/E/Z` ou `/C/F/Z` sont des noms de fichier. On peut remarquer que dans des catalogues différents, on peut utiliser les mêmes identificateurs pour désigner des fichiers ou des catalogues différents.

Un fichier peut être désigné par un nom dit absolu qui permet d'explorer l'arbre à partir du catalogue racine. Il est plus concis et plus efficace de partir de catalogues intermédiaires. Ainsi, chaque utilisateur peut considérer un catalogue comme son catalogue de travail (« working directory ») et désigner les fichiers à partir de ce catalogue. Une convention permet de désigner le catalogue courant (`.` en Unix) et le catalogue père du catalogue courant (`..` en Unix).

Pour faciliter la désignation de fichiers partagés, il est possible d'ajouter des raccourcis de désignation appelés liens.

Exemple 1. Dans la figure 1, les fichiers dont le nom figure dans le catalogue de travail peuvent être désignés simplement par `X` et `Y` ou par `./X` et `./Y`. Le fichier de nom absolu `/C/F/Z` peut aussi être désigné par `../../C/F/Z`

Exemple 2. Sur la figure 2, la flèche allant de `C` à `E` est un lien. Le fichier `/A/E/Z` peut être aussi appelé `/C/E/Z`. Enfin, la structure d'arbre peut être complétée pour permettre l'accès à des volumes amovibles (disquettes ou CD) ou même à des systèmes de fichier situés sur d'autres machines. Une opération appelée montage permet de mettre en correspondance avec une entrée d'un catalogue la racine d'une autre arborescence, par exemple celle qui décrit l'ensemble des fichiers présents sur la disquette qui est chargée dans le lecteur.

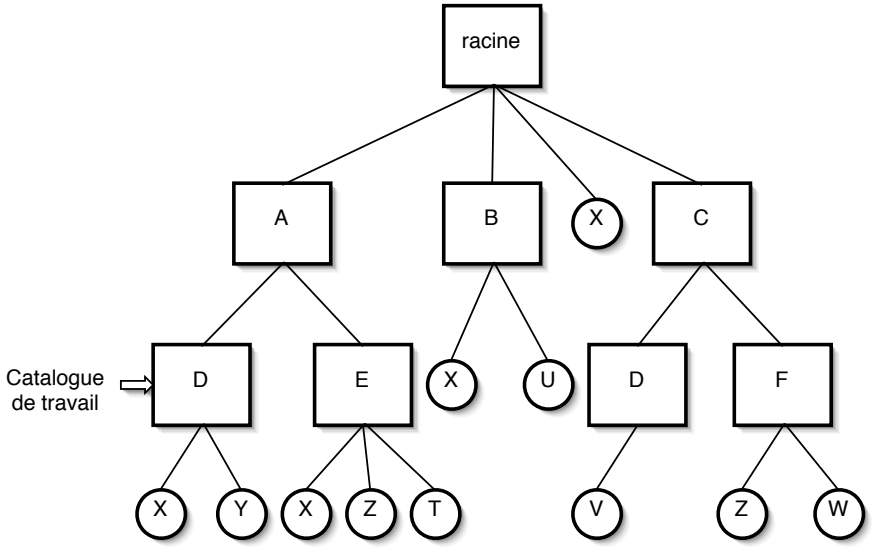

FIG. 1 – Fichiers et catalogues

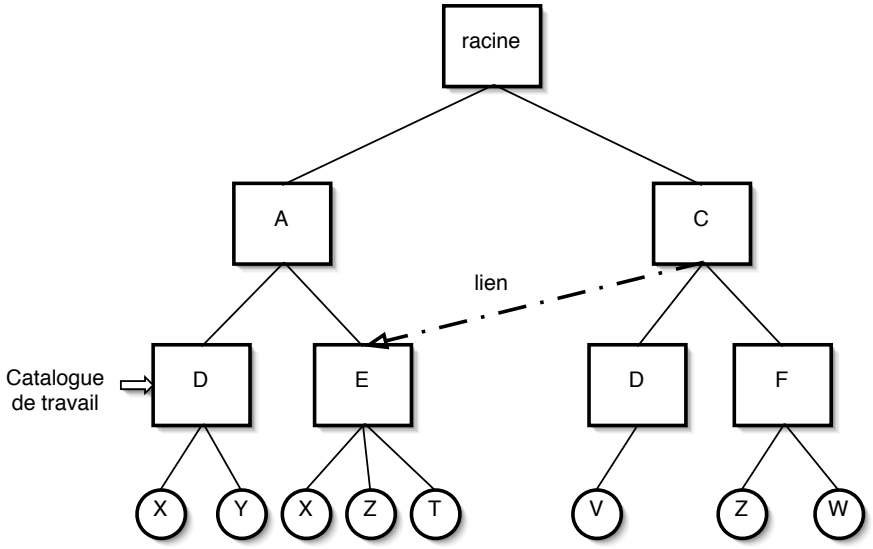

FIG. 2 – Désignation par un lien

4 Réalisation des systèmes de gestion des fichiers

4.1 Représentation des fichiers et des catalogues

4.1.1 Généralités

Pour assurer une conservation d'informations qui persiste en cas de mise hors tension, les objets doivent être conservés sur un support magnétique, en pratique des disques. Les disques sont caractérisés par une grande capacité et par une très forte latence des opérations d'entrée sortie. Cette latence est due aux mouvements du bras et aux temps de rotation ; le rapport entre les latences des accès au disque et à la mémoire principale est de l'ordre de 10^6 ce qui est gigantesque.

En conséquence, les algorithmes de gestion du disque peuvent « gaspiller » de la place ; en revanche, ils doivent limiter dans la mesure du possible le nombre des opérations d'entrée sortie. En particulier, il faut proscrire le parcours de structures chaînées sur disque.

Quant aux fichiers, des mesures ont montré que leurs tailles sont très variables, mais qu'il y a beaucoup de petits fichiers. L'accès séquentiel est beaucoup plus fréquent que l'accès direct, les lectures que les écritures.

4.1.2 Gestion de l'espace disque

Comme pour toute gestion d'espace, la gestion de l'espace disque peut allouer les objets de façon contiguë ou par blocs. La première méthode a l'inconvénient de provoquer un émiettement de l'espace libre ; l'augmentation dynamique de la taille d'un objet est assez compliquée. Nous retenons donc une gestion de l'espace par blocs de taille fixe. Un simple vecteur de bits (1 bit par bloc indiquant si le bloc est libre ou occupé) permet de représenter l'espace libre. La taille de ce vecteur est en général suffisamment réduite (1 M octets pour représenter 30 Go avec des blocs de 4 Ko) pour qu'on puisse en conserver une copie en mémoire pendant les phases de fonctionnement du système.

4.1.3 Représentation des fichiers et des catalogues

Un fichier est constitué d'un ensemble de blocs, éventuellement variable. Si nous étions limités à l'accès séquentiel, ces blocs pourraient tout simplement être chaînés entre eux. Mais un accès direct à un bloc imposerait le parcours de tous les blocs qui le précèdent.

On préfère donc représenter l'espace occupé par un fichier dans une structure auxiliaire, le descripteur du fichier. Ce descripteur contient les attributs du fichier et la liste des blocs du fichier. Un fichier peut alors être désigné par l'adresse ou le numéro de son descripteur.

Un catalogue est un fichier particulier qui a un descripteur comme tous les autres. Un attribut particulier indique qu'il s'agit d'un catalogue. Le contenu d'un catalogue est une suite de doublets (identificateur,numéro de descripteur).

Exemple 1 : les descripteurs d'UNIX V7

voir figure 3

(Le schéma est emprunté à Tanenbaum
http ://www.prenhall.com/divisions/esm/app/author_tanenbaum/custom/mos2e/)

Les blocs d'indirection simples, doubles ou triples permettent la prise en compte de tailles très variables tout en conservant un seul descripteur pour les petits fichiers.

Exemple 2 : les catalogues d'UNIX V7

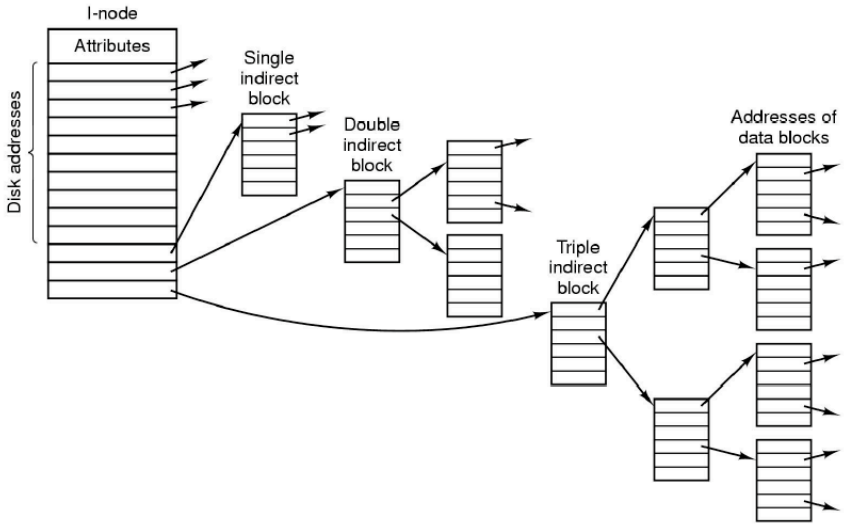

FIG. 3 – Les descripteurs d'Unix V7

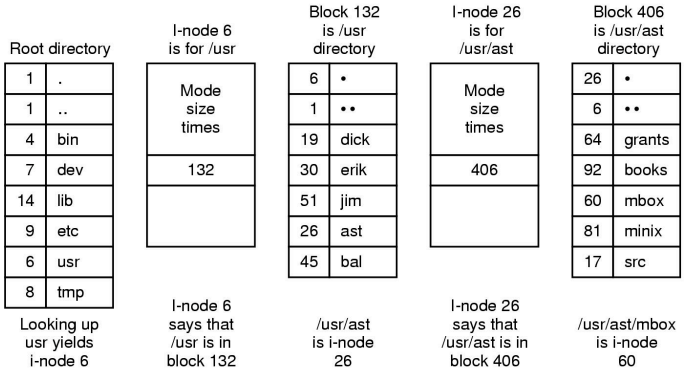

FIG. 4 – Les catalogues d'Unix V7

voir figure 4
http ://www.prenhall.com/divisions/esm/app/author_tanenbaum/custom/mos2e/
Exemple 3 : la représentation des fichiers de PedagOS (cf. 6.2.1)

4.2 Programmation des appels systèmes : principes de base

Un accès à un emplacement d'un fichier implique :
- de localiser sur disque le descripteur de ce fichier,
- de charger en mémoire le descripteur,
- de charger en mémoire le ou les blocs concernés par le transfert,
- de faire une copie en mémoire entre la zone de l'utilisateur et les blocs du fichier (ces deux phases peuvent être faites bloc après bloc),
- dans le cas d'une écriture de fichier, les blocs doivent être recopiés sur le disque.

Pour améliorer l'efficacité, un certain nombre de ces opérations vont être effectuées au moment de l'ouverture du fichier, et leur résultat conservé dans des structures en mémoire centrale. En particulier, le descripteur d'un fichier est chargé en mémoire à l'ouverture. De même, une copie du vecteur décrivant les blocs libres réside en permanence en mémoire centrale, ce qui permet d'accélérer les allocations et libérations de blocs.

Remarque importante. Cette duplication de données entre disque et mémoire centrale présente un inconvénient important : lors d'une modification, la version en mémoire centrale est la version à jour ; en cas de panne, on peut se retrouver avec une version sur disque périmée ou incohérente dans le cas où les structures ont été partiellement recopiées.

4.3 Utilisation d'un « cache » du disque

En général, un programme d'application effectue plusieurs accès à un bloc d'un fichier : les parcours séquentiels sont fréquents ; des fichiers temporaires sont écrits dans une phase d'une application puis relus par la suite, etc. On souhaite regrouper le plus possible les échanges avec le disque concernant un même bloc.

Le principe de la solution est simple : on conserve en mémoire centrale une copie d'un certain nombre de blocs de disque, cet ensemble étant appelé le **cache du disque**. Avant tout transfert, le SGF commence par rechercher si le bloc est déjà dans le cache ; si ce n'est pas le cas, il détermine un bloc à chasser du cache pour y charger le nouveau. La programmation du cache pose deux difficultés, le traitement des écritures et le choix du bloc à effacer.

En l'absence de panne, on pourrait se contenter de recopier un bloc sur le disque au moment où on doit le chasser du cache. Compte tenu des grandes tailles des mémoires, ce sont en pratique des centaines ou des milliers de blocs que le SGF conserve dans le cache. Dans une utilisation individuelle d'un ordinateur, la plupart des fichiers manipulés par une application sont donc en totalité dans le cache. Autrement dit, une panne peut provoquer la perte d'une longue séance de travail. Le SGF doit donc recopier « sans trop tarder » les blocs modifiés, la solution maximaliste consistant à répercuter immédiatement chaque modification sur le disque (écriture immédiate ou « write through »).

Le choix du bloc à effacer est également complexe, car le SGF ne peut pas anticiper sur les futurs accès aux fichiers. On admet en général que les derniers blocs utilisés sont ceux qui ont la plus grande probabilité d'être récupérés. On peut alors chaîner les blocs dans l'ordre de leur utilisation et, quand le SGF a besoin d'un nouveau bloc, il efface le bloc le moins récemment

utilisé. Cet algorithme, appelé LRU pour « least recently used » est d'emploi très fréquent dans les systèmes. Dans le cas du cache du disque, il peut être implanté de façon efficace (il suffit à chaque accès à un bloc de modifier son rang dans une liste). On peut aussi l'améliorer en tenant compte de la nature (fichier, catalogue de travail, catalogue intermédiaire) de l'élément auquel appartient le bloc. Ces améliorations sortent de la portée de ce résumé.

Comme les accès séquentiels sont très fréquents, on peut également utiliser le cache en demandant par anticipation le chargement dans le cache du bloc suivant le bloc courant de chaque fichier.

4.4 Limitation des déplacements du bras

Nous avons vu au chapitre précédent que les mouvements du bras prennent l'essentiel du temps d'entrée sortie sur disque et que le pilote des disques, à son niveau, réordonne les demandes pour minimiser les déplacements du bras.

Les SGF eux-mêmes cherchent à organiser fichiers et descripteurs de façon à diminuer ces mouvements. En premier lieu, les blocs d'un même fichier sont alloués de préférence sur un même cylindre. En second lieu, les descripteurs de fichier ne sont pas regroupés dans une même zone de disque bien séparée de celle des fichiers ; au contraire, on rapproche les descripteurs des blocs de fichier en divisant les disques en groupes de cylindre, chaque groupe ayant sa zone de descripteurs donnant l'implantation des fichiers localisés dans le groupe.

4.5 Fichiers journalisés

La grande taille des caches des disques permet de garantir que la plupart des accès aux fichiers se font sur des blocs présents dans le cache. Dans le cas des lectures, l'accès se limite au cache ; dans le cas des écritures, il faut bien répercuter sur le disque les modifications. Dans les implémentations traditionnelles, les blocs de fichiers résident à des adresses disque fixes et l'écriture d'un ensemble de blocs implique des déplacements du bras pendant lesquels aucun transfert n'a lieu.

L'idée introduite dans les fichiers journalisés (« Log Structured File Systems ») consiste à regrouper les blocs modifiés dans un tampon en mémoire pour les écrire sur disque en une seule opération d'écriture dans des secteurs contigus, c'est à dire avec un seul positionnement du bras. Plus le nombre de blocs écrits est grand, meilleur sera le débit utile du disque. En conséquence, les blocs de fichier changent d'emplacement sur le disque au fur et à mesure des écritures. Le disque est alors utilisé comme une structure séquentielle, d'où le nom de journal (pour journal de bord) qu'on lui donne.

La réalisation des fichiers journalisés pose deux classes de difficultés. Il faut tout d'abord adapter les structures de données pour retrouver les blocs de fichier. L'idée consiste à placer les descripteurs de fichier dans le journal et à terminer chaque écriture par une table donnant la position des descripteurs dans le journal.

En second lieu, le fonctionnement suppose qu'il existe sur le disque une zone libre assez grande pour y effectuer l'ensemble des écritures. Comme les écritures se font de façon séquentielle, la fin du disque sera inévitablement atteinte. Il faut alors réorganiser l'ensemble du disque pour regrouper dans une seule zone l'ensemble des blocs encore valides et utilisés et obtenir ainsi une seule zone libre de grande taille. C'est cette opération de réorganisation, coûteuse et complexe, qui est le facteur limitant.

Les fichiers journalisés se révèlent meilleurs que les meilleurs SGF traditionnels pour un taux de remplissage des disques faible ; ils sont à peu près au même niveau lorsque le taux de remplissage atteint 80 % environ.

Comme les disques sont en pratique souvent pleins, les fichiers journalisés n'ont pas connu la généralisation attendue il y a quelques années.

5 Fiabilité

L'objectif premier des SGF est d'assurer la conservation permanente des informations. Il faut donc développer des techniques permettant de conserver les informations en présence de pannes ou simplement de vieillissement des supports. Les SGF doivent également offrir des garanties de confidentialité et de protection contre les modifications intempestives, malveillantes ou non.

5.1 Résistance aux pannes

La seule façon de résister à des défaillances, qu'elles soient matérielles ou logicielles, est de conserver des informations de façon redondante. Cette redondance peut s'envisager à plusieurs niveaux :

- au niveau des disques eux mêmes, avec des techniques RAID qui permettent de reconstituer des informations devenues inaccessibles,
- au niveau des structures de données sur disque, pour pouvoir reconstituer en cas d'effacement partiel d'un disque le vecteur des blocs libres ou les différents catalogues ou descripteurs de fichier,
- au niveau des fichiers par l'exécution régulière de copies de sauvegarde.

Développons un peu ce dernier point que l'augmentation des tailles des disques rend de plus en plus délicat. Quelques remarques pour commencer. Quel que soit le support d'une copie de sauvegarde, même s'il est jugé permanent comme un CDrom, il se dégrade avec le temps et devient illisible. La conservation des informations implique la recopie périodique sur de nouveaux supports. La sécurité absolue n'existe pas et toute mesure de résistance aux défaillances se traduit par un coût (coût de stockage des informations, coût des recopies, etc.). C'est à chaque responsable informatique de définir une politique de sécurité adaptée aux informations qu'il gère. Les principales techniques à combiner sont les suivantes.

- Dupliquer toutes les opérations d'écriture sur des supports distincts, éventuellement éloignés. On peut par exemple effectuer les écritures en parallèle sur deux disques aux contenus identiques. On peut aussi effectuer les écritures en parallèle sur le disque support des fichiers et sur un autre support géré séquentiellement et appelé le journal. Cette dernière technique est très employée dans les SGF modernes ; elle est appelée en anglais « journalized file systems ».
- Sauvegarder périodiquement l'ensemble des fichiers modifiés depuis la dernière sauvegarde.
- Sauvegarder périodiquement l'ensemble des fichiers.

5.2 Protection des fichiers

On regroupe sous ce terme l'ensemble des techniques permettant de contrôler la validité des opérations déclenchées sur des fichiers . L'étude générale des techniques de protection sort du

cadre des fichiers et sera reprise ultérieurement.

Dans les SGF les plus courants, la protection repose d'abord sur l'identification de l'utilisateur qui effectue les accès (phase d'authentification). Pour chaque fichier, son propriétaire (celui qui a créé le fichier) doit définir les utilisateurs qui ont le droit d'y accéder, et pour quelles opérations. Cette liste d'utilisateurs et de droits est stockée au niveau des attributs de chaque fichier et catalogue. Les utilisateurs sont souvent regroupés dans des classes de mêmes droits pour rendre la représentation plus compacte.

Exemple. En UNIX, les utilisateurs sont rassemblés dans des groupes. Les accès élémentaires à un fichier sont la lecture, l'écriture et l'exécution. Le propriétaire d'un fichier définit les droits qu'il s'accorde sur le fichier, ceux qu'il accorde aux membres de son groupe et ceux fournis aux autres utilisateurs. Un vecteur de 9 bits permet de représenter ces droits. Quand un processus s'exécute, on lui associe l'identification UID de l'utilisateur qui l'a créé ainsi que l'identification GID du groupe de cet utilisateur. A chaque ouverture de fichier, le SGF vérifie que le mode d'ouverture est compatible avec les droits associés au couple (UID,GID) figurant dans les attributs.

6 Le SGF de PedagOS

6.1 Interface utilisateur

Le système de fichiers de PedagOS est très inspiré de celui d'Unix. Les fichiers sont des suites d'octets non structurées. Un système de catalogues arborescent permet à un processus de désigner un fichier soit par un nom global, soit à partir du catalogue courant. La désignation du catalogue courant est contenue dans le descripteur de chaque processus (champ `i_df_cat_courant`). Dans le programme d'un processus, un fichier est désigné par un indice dans une table de descripteurs de fichiers ouverts. Cette table fait partie du descripteur de processus (champ `dfo[MAX_FO_PROC]`). Le descripteur de fichier ouvert est alloué lors de l'opération d'ouverture ; il est libéré à la fermeture du fichier.

6.1.1 Créations, destructions, changements de taille

Le SGF doit permettre de créer ou détruire des fichiers ou des catalogues, ou d'augmenter dynamiquement leur taille. Nous laissons de côté ces aspects ici, mais les structures introduites permettent leur réalisation.

6.1.2 Ouverture-fermeture

A l'ouverture d'un fichier par un processus, l'appelant fournit un nom symbolique absolu ou relatif et l'appel système lui retourne un numéro de descripteur de fichier ouvert ou un code d'erreur. Le numéro de descripteur sera utilisé comme désignation locale au processus pour toute la phase pendant laquelle le fichier reste ouvert. C'est à l'ouverture que sont effectuées toutes les vérifications nécessaires (nom correspondant à un fichier inexistant par exemple) et que sont construites les différentes tables permettant de réaliser les accès. L'appel système correspondant est

```
ouvrir (nom_fich,mode) -> i_dfo
```

Lorsque l'ouverture se passe bien, `ouvrir` retourne un indice de descripteur de fichier ouvert. En cas d'erreur, `ouvrir` retourne un code d'erreur négatif avec le sens suivant :

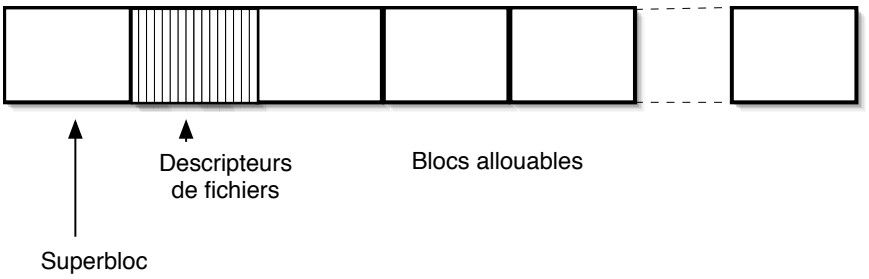

FIG. 5 – Structure du disque

-1 on devrait avoir un catalogue et on traverse un fichier,
-2 nom inexistant dans un catalogue,
-3 trop de fichiers ouverts dans le système,
-4 trop de fichiers ouverts par le processus.

L'appel système de fermeture reçoit en paramètre un numéro de descripteur de fichier ouvert. Les tables construites à l'ouverture sont détruites. L'appel système est

fermer(i_dfo) -> code de retour

La procédure retourne 0 si l'opération s'est bien exécutée, -1 si le descripteur est invalide, -2 s'il ne correspond pas à un fichier ouvert.

6.1.3 Lecture écriture

Un fichier ouvert possède une position courante, fixée par défaut au début du fichier lors de l'ouverture. Les opérations de lecture et d'écriture se font à partir de la position courante qui sera augmentée en fin d'opération du nombre d'octets transférés. Un appel système permet de modifier explicitement la position courante, ce qui permet de réaliser des accès directs. Les deux opérations

```
lire (i_dfo, adr_mem, nbcar) -> code de retour
ecrire (i_dfo, adr_mem, nbcar) -> code de retour
```

transfèrent `nbcar` octets à partir de la position courante du fichier `dfo` soit vers la zone utilisateur d'adresse `adr_mem` (ordre `lire`), soit à partir de cette zone (ordre `ecrire`). Le code de retour est 0 si l'opération s'est bien passée, et sinon un code d'erreur :

-1 descripteur invalide
-2 ordre incompatible avec le mode d'ouverture
-3 dépassement de la taille logique (lecture) ou physique (écriture)

L'appel système

```
fixer_position(i_dfo,position) -> code de retour
```

permet de changer la position courante du fichier, le code de retour pouvant être -1 ou -3.

L'appel système

```
obtenir_taille (i_dfo) -> taille
```

permet d'obtenir la taille logique du fichier ou -1 en cas de descripteur invalide.

6.2 Réalisation

6.2.1 Implantation sur disque

La figure 5 montre l'utilisation de l'espace disque. Il est organisé comme un ensemble de blocs de taille fixe. Un fichier (ou un catalogue) occupe un ensemble de blocs non contigus. Une table appelée descripteur de fichier ou `df` permet de repérer tous les blocs d'un fichier. Tous les `df` sont regroupés dans une zone particulière du disque et de façon interne au système, un fichier (ou un catalogue) est désigné par un numéro `i_df` de descripteur de fichier `df`. Des tables présentes dans une zone fixe permettent de conserver la liste des blocs libres et la liste des `df` libres. Il s'agit de vecteurs de bits.

Nous utilisons ces tables par les procédures

```
allouer_bloc_disque -> ibloc
liberer_bloc_disque (ib)
allouer_df -> i_df
liberer_df (i_df)
```

Structure du disque

Le superbloc contient les vecteurs d'allocation des blocs et des descripteurs de fichier. On trouve ensuite les blocs contenant les descripteurs de fichiers, puis les blocs disponibles pour les fichiers et les catalogues.

Composition d'un descripteur de fichier

Un `df` est représenté sur disque comme une structure de taille fixe qui contient les informations suivantes :

```
struct df {
   nature ;        // fichier ou catalogue
   taille_phy ;    //taille physique en octets
   taille_log ;    // déplacement du dernier caractère écrit
   blocs [NBLOCS] ;  //numéros des blocs
   bloc_indirect ;       //numéro du bloc d'indirection simple
   }
```

Lorsqu'un fichier occupe moins de `NBLOCS` blocs, le tableau `blocs` suffit à contenir la liste des blocs du fichier. Pour les fichiers plus grands, on utilise `bloc_indirect` qui contient le numéro d'un bloc qui est utilisé pour contenir les numéros des blocs suivants du fichier.

Catalogues

Un catalogue est un fichier particulier dont le contenu est une suite de doublets `(identificateur,numéro de df)`.

Les deux premiers doublets correspondent au catalogue lui-même (entrée `.`) et au catalogue père (entrée `..`). La taille logique présente dans le `df` d'un catalogue permet de comptabiliser le nombre d'entrées.

6.2.2 Accès au disque

Un transfert entre disque et mémoire concerne toujours un bloc entier. Lorsqu'une copie d'un bloc disque est présente en mémoire, son contenu est décrit par un descripteur de bloc en mémoire

ou dbm dont le format sera précisé plus loin.

La procédure

obtenir_bloc (bloc_disque) -> dbm

permet de récupérer en mémoire centrale un bloc de disque.

La procédure

liberer_bloc (dbm)

indique la fin de l'utilisation d'un bloc par un processus.

Pour améliorer l'efficacité, les derniers blocs accédés, qu'il s'agisse de blocs contenant des df ou des blocs de fichiers ou de catalogues, sont conservés en mémoire centrale dans un cache. Ce cache contient TAILLE_CACHE blocs. Un bloc reste dans le cache tant qu'on n'a pas besoin de récupérer sa place pour charger un nouveau bloc.

Le descripteur de bloc en mémoire a la structure suivante :

```
struct dbm {
   cpt ;           //nombre d'utilisateurs du bloc
   ad_disque ;     // adresse du bloc sur disque
   ad_mem ;        // adresse du bloc en mémoire centrale
   etat ;             // intact ou modifié
   liens ;         // chaînage dans une liste LRU
   }
```

Le champ cpt compte le nombre des processus qui utilisent le bloc.

Le champ etat permet de savoir si un bloc en mémoire a été modifié depuis son chargement. Si c'est le cas, il devra être recopié sur disque au plus tard quand le système aura besoin de récupérer sa place en mémoire pour charger un nouveau bloc.

Le champ liens est utilisé par les procédures de gestion du cache pour récupérer un emplacement de bloc. En général, on supprime de la mémoire le bloc qui a été le moins récemment utilisé (« least recently used » ou LRU).

ATTENTION : LE PILOTE DONNE AU C4 EST A REVOIR

La procédure obtenir_bloc consulte et met à jour le cache par l'intermédiaire des procédures

recherche_en_cache (bloc_disque) -> dbm

qui retourne un descripteur de bloc en mémoire centrale (NULL si le bloc n'est pas en mémoire) et

trouve_place_cache -> dbm

qui retourne un descripteur de bloc utilisable.

La procédure liberer_bloc se limite à diminuer de 1 le compteur dbm->cpt, ce sont les algorithmes de gestion du cache qui se chargeront du vidage éventuel sur disque.

PROGRAMMATION DU CACHE ? ? ? ?

6.2.3 Structures de données en mémoire centrale

Pour limiter les accès au disque, le système de gestion des fichiers construit un certain nombre de tables en mémoire centrale. On recopie tout d'abord au démarrage du système les vecteurs de disponibilité des blocs libres et des df libres. Lorsque le SGF a besoin de consulter un descripteur de fichier, il le charge en mémoire dans une structure appelée descripteur de fichier en mémoire ou

dfm. Un dfm contient une copie du descripteur de fichier et quelques champs complémentaires. Il a la structure suivante :

```
struct dfm {
   // copie en mémoire de df
   nature ;
   taille_phy ;
   taille_log ;
   blocs[NBLOCS] ;
   bloc_indirect ;
   // champs supplémentaires
   i_df ;       // numéro du df sur disque
   cpt ;        // nombre d'utilisateurs du dfm
   etat ;       // df intact ou modifié depuis la création du dfm
   }
```

Une table appelée table des descripteurs de fichier en mémoire ou tdfm rassemble l'ensemble des dfm. Cette table est un simple tableau de pointeurs vers les dfm chargés (ou NULL si l'entrée est disponible). On notera i_tdfm l'indice d'une entrée dans cette table. Les procédures de gestion de la table sont :

obtenir_dfm (i_df) -> i_tdfm

qui recherche tout d'abord si le descripteur i_df est présent dans la table, et sinon le charge à partir du disque, et

liberer_dfm(i_tdfm).

La tdfm doit être assez grande pour contenir les dfm de tous les fichiers en cours d'utilisation (c'est à dire ouverts). Un descripteur de fichier en mémoire n'est toutefois pas effacé à la fermeture : on le conserve en mémoire jusqu'à ce que le SGF ait besoin de récupérer sa place pour charger un nouveau dfm.

Lorsqu'un fichier est ouvert, le SGF doit conserver le mode (lecture,écriture ou lecture-écriture) d'ouverture du fichier ainsi que l'indice de la position courante dans le fichier. Ces informations sont regroupées dans une table globale des fichiers ouverts ou tgfo (indice i_tgfo). Une entrée dans cette table a la structure suivante.

```
struct entree_tgfo {
   cpt ;        // compteur des ouvertures en cours
   mode ;       // mode d'ouverture
   position ;      // position courante
   i_tdfm ;     // localisation du descripteur de fichier en
      mémoire
   }
```

Enfin, pour permettre à chaque processus de désigner ses fichiers ouverts indépendamment des autres, ainsi que pour permettre les redirections des entrées sorties standard vers des fichiers, on dispose pour chaque processus d'une table de descripteurs de fichiers ouverts dfo. L'entrée dfo[i_dfo] contient simplement l'indice i_tgfo du fichier désigné par i_dfo dans la table globale des fichiers ouverts.

La figure 6 résume les structures de données en mémoire.

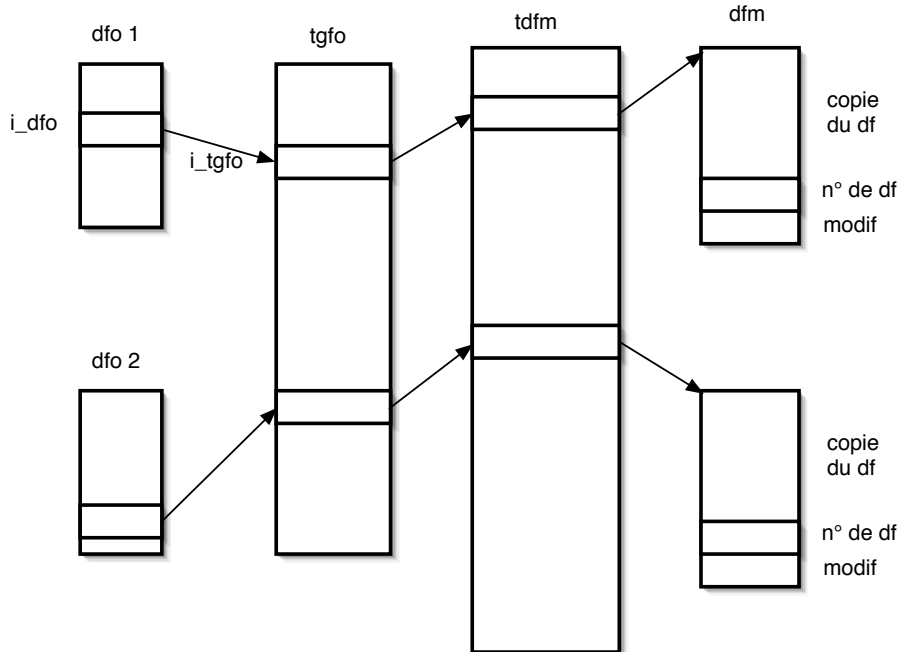

FIG. 6 – Les tables du SGF

6.2.4 Implantation de quelques appels système

Hypothèses

Nous allons donner l'ossature des traitements de quelques appels système représentatifs de la programmation du SGF et de l'utilisation des structures de données introduites. **Nous nous limitons pour des raisons de concision à des accès à des fichiers existants sans modification de leur taille**. Nous allons examiner trois appels système de complexité croissante permettant de changer la position courante dans un fichier ouvert, de lire une suite de caractères dans un fichier et d'ouvrir un fichier.

Les deux derniers de ces appels système peuvent imposer un ou plusieurs blocages du processus appelant pour effectuer des échanges avec le disque. Nous admettons que la gestion des processus permet ce blocage en mode maître au cours du traitement d'un appel système, avec au déblocage le retour au contexte d'exécution préalable au blocage. Notez bien que le noyau fourni au chapitre 3 n'offre pas cette possibilité. Nous en traiterons au chapitre 6 décrivant la structure globale d'un système d'exploitation simple.

Changement de la position courante dans un fichier ouvert

L'appel système de changement de la position courante a deux paramètres, l'indice dans le processus d'un descripteur de fichier ouvert et la position finale. Il peut échouer si le descripteur fourni est invalide ou si la nouvelle position est en dehors du fichier.

```
exec_fixer_position(i_dfo,position) {
    if ((i_dfo < 0) || (i_dfo >= MAX_FO_PROC))
        return (-1) ; // mauvaise valeur d'i_dfo
```

```
    if ( proelu->dfo [i_dfo] < 0) return (-1) ; /* fichier non
       ouvert */
    i_tgfo = proelu->dfo [i_dfo] ;  /* indice du fichier dans la
       tgfo */
    dfm= tdfm [tgfo[i_tgfo]->i_tdfm ] ; // pointeur vers le dfm
    if ((position > dfm->taille_log) || (position <0))
       return (-3) ; //  nouvelle position invalide
    tgfo[i_tgfo]->position = position ; /* nouvelle position dans
       la tgfo */
    return (0) ;   // appel système réussi
    }
```

Appel système de lecture `lire(i_dfo,adr_mem,nbcar)`

Dans le cas d'un appel système valide, nous devons transférer les caractères du fichier vers la zone utilisateur. Ces caractères se trouvent dans un ou plusieurs blocs du fichier qu'il faut localiser sur le disque à l'aide du `dfm`, puis obtenir dans le cache. Une fois un bloc dans le cache, la portion du bloc concerné doit être recopiée dans le tampon utilisateur. Le transfert peut faire intervenir plusieurs blocs.

Nous introduisons une procédure intermédiaire

`logique_physique (dfm,i_bloc) -> adr_disque`

qui permet d'obtenir l'adresse sur disque d'un bloc à partir de son numéro logique dans le fichier. Cette procédure doit éventuellement charger en mémoire le bloc d'indirection.

La procédure lire peut échouer si le descripteur est invalide ou si elle tente de lire des caractères au-delà de la fin logique du fichier (attention, cette spécification est différente du read d'UNIX pour le traitement des fins de fichier).

```
exec_lire(i_dfo,adr_mem,nbcar) {
    if ((i_dfo < 0) || (i_dfo >= MAX_FO_PROC))
       return (-1) ; // mauvaise valeur d'i_dfo
    if ( proelu->dfo [i_dfo] < 0) return (-1) ; /* fichier non
       ouvert*/

    i_tgfo = proelu->dfo[i_dfo] ;  /* indice du fichier dans la
       tgfo */
    if (tgfo[i_tgfo]->mode != lecture)  return(-2) ; /* non
       lecture */
    dfm= tdfm [tgfo[i_tgfo]->i_tdfm ] ; // pointeur vers le dfm
    pos = tgfo[i_tgfo]->position
    if (pos+nbcar  > dfm->taille_log)
       return (-3) ;      // tentative de lecture après la fin
    /* bl bloc logique , depl déplacement dans le bloc , ind index
       zone utilisateur
TAILLEBLOC est la taille du bloc sur disque */
    bl = pos / TAILLEBLOC ; depl = pos % TAILLEBLOC ;
    ind = 0 ;
```

```
/* la boucle suivante amène les blocs dans le cache, puis fait la
    copie en zone utilsateur  */

while (nbcar >0 ) {
   bdisque = logique_physique (dfm,bl) ;
   dbm = obtenir_bloc (bdisque) ; /* transfert du bloc dans le
      cache */
   // nb de caractères à transférer de ce bloc
   long = min (nbcar, TAILLEBLOC-depl) ;
   copie_syst_utilisateur (dbm->adr_mem+depl,  //adr système
                     adr_mem+ind,  // adr utilisateur
                     proelu, long) ;
   liberer_bloc(dbm) ; /*le bloc n'est plus utilisé pour cet
      appel */
   bl = bl+1 ; depl = 0 ; ind = ind + long ;
   nbcar = nbcar-long ;
   }   // end while

// mise à jour position dans tgfo ; on a transféré ind caractères
tgfo[i_tgfo]->position = tgfo[i_tgfo]->position + ind ;
return (0) ; // transfert réussi
}
```

Ouverture de fichier

Définissons tout d'abord une procédure auxiliaire
`recherche_arbre (nom,i_df) -> i_df`
dans laquelle `nom` est un nom de fichier (suite d'identificateurs séparés par des /) et `i_df` l'indice du descripteur du catalogue de départ de la recherche. En cas de succès, `recherche_arbre` retourne l'indice du descripteur du fichier `nom` ; en cas d'erreur, elle retourne un code d'erreur négatif

- -1 fichier au lieu d'un catalogue
- -2 nom inexistant dans un catalogue

Par exemple, soit le nom `/un/deux` et `i_d` le numéro de descripteur du catalogue racine. La procédure `recherche_arbre` commence par installer en mémoire un `dfm` pour ce catalogue, puis cherche dans ce catalogue la chaîne `un`. En cas de succès, la recherche fournit le `df` du catalogue de nom `un`, dans lequel la procédure va chercher le nom `deux`. En cas de succès, elle retourne le numéro de descripteur du fichier `deux`.

```
recherche_arbre(nom,i_df) {
   while ( !fin_de_nom) {
      i_tdfm= obtenir_dfm(idf) ;
      dfm= tdfm[i_tdfm] ;
      if (dfm->nature != catalogue)   /* on ne recherche pas en
         fichier */
```

```
        { liberer_dfm(i_tdfm) ; return (-1) ; } ;
      ch= identificateur_suivant ;
      i_df = recherche_catal (ch,dfm) ; /* explore le catalogue
         courant */
      liberer_dfm(i_tdfm) ;   /* on n'a plus besoin de ce
         catalogue */
      if (i_df<0) return (-2) ; // identificateur inexistant
      // sinon poursuite du while avec l'identificateur suivant
      } ;
/* on a exploré tous les champs, i_df est le numéro de
   descripteur à envoyer */
   return (i_df) ;
   }
```

Passons maintenant à l'ouverture proprement dite. Le catalogue initial de recherche est soit le catalogue racine, soit le catalogue courant du processus faisant l'appel (proelu). Lorsque `recherche_arbre` réussit, on alloue des entrées dans les tables `dfo` et `tgfo` ; on construit un `dfm` pour le fichier dont `recherche_arbre` a renvoyé le numéro de descripteur ; il reste alors à initialiser les tables.

```
exec_ouvrir (nom,mode) {
   if (nom[0]='/') i_df = i_df_racine ;
   else i_df = proelu->i_df_cat_courant ;
   indice_df =recherche_arbre(nom,i_df) ;
   if (indice_df<0) return(indice_df) ;
   /*on va maintenant construire les différentes tables décrivant
       le fichier */
   i_tgfo = allouer_tgfo() ; // recherche emplacement dans tgfo
   if (i_tgfo<0) return (-3) ; //tgfo pleine
   i_dfo = allouer_dfo(proelu->dfo) ;
   if (i_dfo<0)   //trop de fichiers ouverts par le processus
      {liberer_tgfo(i_tgfo) ; return (-4) ;} ;
   proelu->dfo[i_dfo] = i_tgfo ;
   tgfo[i_tgfo]->cpt = 1 ;
   tgfo[i_tgfo]->mode = mode ;
   tgfo[i_tgfo]->position = 0 ; // début du fichier
   i_tdfm = obtenir_dfm(indice_df) ; //construction du dfm
   ++tdfm[i_tdfm]->cpt ; // augmentation compteur d'utilisation
   tgfo[i_tgfo]->i_tdfm = i_tdfm ;
   return(i_dfo) ;
   }
```