Cours Système n°2: les fichiers 1/2

Éric Gaudefroy

École Hexagone
Cursus Cyberdéfense - M1

Introduction

L'organisation des fichiers tient une place essentielle dans les systèmes d'exploitation. En particulier, dans les systèmes UNIX, il est parfois dit que tout est fichier. En effet, en plus des fichiers standards contenant des données, il existe de nombreux fichiers spéciaux ; Certains permettent de représenter les divers périphériques [1], d'autres sont des pseudo-fichiers contenant de précieuses informations sur la plate-forme, d'autres encore représentent les *sockets* ou les *pipes*, etc...

Ces fichiers s'organisent en systèmes de fichiers, qui sont de natures diverses. Dans un système UNIX, tout système de fichier actif s'intègre dans le VFS (*Virtual File System*) par l'opération de montage. Ce premier cours sur les fichiers a pour objectif de présenter le fonctionnement des systèmes de fichiers classiques, et leur stockage sur des supports de stockage. Les attributs des fichiers seront ensuite détaillées, ainsi que la représentation utilisée par le noyau pour la gestion des fichiers. Enfin, les primitives permettant de manipuler les répertoires et les fichiers réguliers, ainsi que les droits associés, seront présentées.

1. Périphériques de stockage

1.1. Types de périphériques

Les données visibles au niveau d'une machine peuvent être issues de différents types de supports, typiquement :

- des supports physiques, amovibles (disquettes, CD-ROM, clés USB, etc...) ou non (disques durs) ;
- des supports réseau (partages NFS, Samba [2], etc...) ;
- des supports virtuels (disques virtuels, *procfs*, etc...).

1.2. Disques durs

Les disques durs sont composés de secteurs, qui historiquement étaient regroupés en têtes et en cylindres [3]. Ce découpage, visible depuis le système d'exploitation, n'a plus aujourd'hui de réalité

1. Les interfaces réseau font cependant exception à cette règle.
2. Samba est l'implémentation libre du protocole SMB/CIFS, correspondant au « Voisinage Réseau » du système Windows.
3. On parle d'adressage CHS, pour Cylinders, Heads, Sectors

physique sur le disque. Aussi, l'adressage des secteurs se fait désormais selon le mode LBA (Logical Block Adressing), c'est-à-dire un simple compteur qui énumère l'ensemble des secteurs de 0 à N-1.

Les accès se font via un contrôleur disque, qui interprète les commandes et les adresses. Il peut de plus choisir d'optimiser les accès (à l'aide d'un cache matériel) afin de minimiser le déplacement des têtes de lecture. La technologie SMART (*Self Monitoring Analysis and Reporting Technology*) embarquée dans les disques offrent certaines fonctions d'analayse et de réparation du disque dur. Elle gère en particulier les redirections matérielles des secteurs défectueux.

Remarque : la « récupération » des secteurs défectueux par le contrôleur matériel se fait à l'aide de listes de correspondance entre les secteurs virtuels et les secteurs réels. Il existe également d'autres fonctions, telles que la HPA (*Host Protected Area*), une zone utilisable pour stocker une partition de secours du système d'exploitation, ou le DCO (*Device Configuration Overlay*) qui masque certaines parties du disque vis-à-vis des accès standards réalisés par l'OS.

Tous ces éléments montrent que la représentation logique des secteurs peut être assez éloignée de la réalité du fonctionnement sur les plateaux du disque dur. Ces divers mécanismes sont à prendre en compte lors de l'étude de la problématique d'effacement logique d'un disque dur.

Les premiers octets d'un disque dur forment le *Master Boot Record* (MBR). Le MBR contient des méta-informations relatives au disque, et notamment :

- du code exécutable pour démarrer la machine (si le BIOS choisit de démarrer sur ce périphérique) ;
- une table de partitions DOS/fdisk (facultatif sur des systèmes UNIX purs) ;
- un « nombre magique » permettant de vérifier la validité du MBR.

Le détail du MBR est donné dans les tableaux 1 et 2. Les partitions DOS/fdisk sont au nombre de quatre, dites partitions primaires. Une des partitions primaires peut optionnellement être subdivisée en une ou plusieurs partitions étendues, commençant chacune par un *Extended Boot Record* (EBR) contenant une sous-table de partitions. Les partitions sont repérées par un numéro de type. Des exemples sont donnés dans le tableau 3. Sous de nombreux systèmes d'exploitation, la manipulation des partitions s'effectue par la commande **fdisk**, ou une de ses variantes **cfdisk**, **sfdisk**...

Offset	**Taille (en octets)**	**Champ**
0x000	440	Zone réservée au code
0x1B8	4	Signature optionnelle du disque
0x1BC	2	Réservé (généralement deux octets nuls)
0x1BE	16	Premier enregistrement de partition
0x1CE	16	Second enregistrement de partition
0x1DE	16	Troisième enregistrement de partition
0x1EE	16	Quatrième enregistrement de partition
0x1FE	2	Signature du MBR (AA55 en little endian)

TABLE 1 – Description du contenu du MBR. Détail des enregistrements dans le tableau 2.

Offset	Taille (en octets)	Champ
0x00	1	État de la partition : 0x80 signifie que la partition est amorçable, sinon le champ doit valoir 0x00
0x01	3	Adresse au format CHS du début de la partition
0x04	1	Type de partition (quelques exemples des types disponibles sont présentés dans le tableau 3)
0x05	3	Adresse au format CHS de la fin de la partition
0x08	4	Numéro du premier secteur de la partition (format LBA)
0x0c	16	Taille de la partition en secteurs (format LBA)

TABLE 2 – Détail d'un des quatre enregistrement de la table des partitions, située en fin de MBR.

Id.	Type
0x00	Partition non affectée
0x07	Partition HPFS / NTFS (Windows NTx)
0x0c	Partition FAT 32 (Windows, utilisé pour les clés USB)
0x05, 0x0f, 0x85	Méta-partition contenant des partitions étendues
0x82	Partition d'échange (swap) Linux
0x83	Partition ext3 Linux
0xa5	Méta-partition FreeBSD
0xa6	Méta-partition OpenBSD
0xa9	Méta-partition NetBSD
0xfd	Partition RAID Linux

TABLE 3 – Quelques exemples de types de partitions.

1.3. Gestion des disques sous Linux

Comme la plupart des périphériques, les disques sont vus comme des fichiers spéciaux :

- disques IDE : /dev/hda (maître nappe 1), /dev/hdb (esclave nappe 1), /dev/hdc (maître nappe 2), /dev/hdd (esclave nappe 2) ;
- disques SCSI ou SATA, projection des unités de stockage USB : /dev/sda, /dev/sdb, etc... ;
- partitions primaires (ex : /dev/sda) : /dev/sda1 à /dev/sda4 ;
- partitions étendues : /dev/sda5, etc... ;
- lecteurs de disquettes : /dev/fd0, etc... ;
- lecteurs de disques optiques : /dev/cdrom (lien vers le vrai périphérique), /dev/sr0, etc...

La copie bas niveau d'un disque ou d'une partition s'effectue au moyen de la commande **dd** :

```
dd if=<src> of=<dst> [bs=<taille_blocs>] [count=<n_blocs>]
[skip=<blocs_src>] [seek=<blocs_dst>]
```

Elle permet de copier <n_blocs>*<taille_blocs> octets (par défaut, copie complète) en sautant si nécessaire des blocs dans la source (skip) et/ou la destination (seek). La source et la cible peuvent représenter des disques (/dev/hda, /dev/sdb1, /dev/cdrom) ou des fichiers normaux (/tmp/cd_img.iso).

Remarque : Sous Windows, C : représente par défaut la première partition que le système sait gérer (FAT ou NTFS), D : la deuxième, etc...

2. Systèmes de fichiers

Un système de fichier (*filesystem* en anglais) est une représentation permettant l'organisation des fichiers. Les systèmes de fichiers sont adaptés à chaque type de support :

- disque dur : FAT32 (pauvre, mais très portable), NTFS (pour Windows), EXT2/3/4 (pour Linux), UFS (pour BSD) ;
- CD-ROM : ISO9660 (adapté aux accès en lecture seule).

La bonne connaissance d'un système de fichiers permet notamment de :

- faciliter les opérations de réparation manuelle en cas de sinistre (perte de données importantes) ;
- analyser les possibilités de stockage illicite d'information sur un disque (blocs non utilisés, etc...) ;
- rechercher des preuves suite à la saisie d'une machine dont les fichiers ont été effacés ;
- envisager d'ajouter son support sur un OS qui ne sait pas encore le gérer.

Voici quelques éléments intervenant lors de la définition d'un système de fichiers (liste non exhaustive) :

- optimisation des accès en lecture/écriture ;
- optimisation pour les petits ou les gros fichiers ;
- prise en compte de la richesse des attributs manipulés par l'OS (permissions, fichiers spéciaux, etc...) ;
- minimisation de l'espace inutilisable pour stocker les données des fichiers ;
- facilitation de la reprise sur erreur (systèmes journalisés).

Sur un disque dur, un système de fichiers occupe une partition (primaire ou étendue). Les systèmes de fichiers sont généralement composés de :

- une zone de boot, similaire au MBR (facultatif) ;
- un super-bloc contenant les paramètres du système de fichiers (taille, etc...) ;
- une zone de méta-données sur les fichiers présents (table d'allocation, etc...) ;
- une série de blocs de données (contenus de fichiers).

Nota : le système de fichiers EXT est abordé plus en détails dans le cours suivant.

2.1. Généralités sur la représentation logique

Chaque système de fichiers (FS) contient un répertoire racine, des sous-répertoires, des fichiers de données. Ces différents éléments sont dotés d'attributs dont le détail dépend du FS. Le système d'exploitation réalise une arborescence globale unifiant les FS importés, en traduisant les attributs spécifiques de chaque FS en attributs natifs de l'OS. Sous UNIX, l'OS gère un système de fichiers virtuel (VFS). Un des systèmes de fichiers joue le rôle de racine (/), les autres viennent se greffer dans les sous-répertoires. Les commandes **mount** et **umount** permettent d'accrocher et de décrocher un FS dans l'arborescence VFS courante :

```
mount [-t<type>] /dev/<partition> <repertoire_montage>
umount /dev/<partition> (ou umount <repertoire_monte>)

#Exemple
mount -t vfat /dev/sdb1 /mnt/usb
```

Le montage d'un système de fichiers consiste à fusionner en mémoire le répertoire d'accueil, qui conserve son père, avec le répertoire racine du FS monté, qui apporte ses fils. Il est paramétrable par des options (cf. **man mount**), par exemple pour interdire toute modification du FS indépendamment des permissions sur ses fichiers. Le démontage n'est possible que si aucun fichier du FS n'est en cours d'utilisation et provoque une synchronisation des caches d'entrée/sortie du VFS qui lui sont associés.

La commande **df** permet d'affiher la liste des systèmes de fichiers non virtuels montés et leurs taux de remplissage. **mount** (sans argument) fournit la liste complète des montages actifs et des options associées.

Remarque : Sous Linux, le pseudo-fichier /proc/mounts montre la liste des montages effectivement gérés par le noyau. Son contenu est donc plus fiable que les indications fournies par la commande **mount**.

Sous Windows, on peut voir le « Poste de travail » comme l'équivalent de la racine. Les montages se font par la reconnaissance de partitions et la « Connexion de lecteurs réseau » . Le « Panneau de configuration » est un système de fichiers virtuel.

2.2. Arborescences

Les répertoires et fichiers reconnus par le système d'exploitation forment une arborescence. Pour un système donné, de nombreux répertoires ont un usage particulier. Ainsi, les principaux exécutables du système sont situés dans **/bin** sous Unix et dans **C :\Windows**) et son sous-répertoire system32 sous Windows. L'organisation des répertoires UNIX est détaillée dans la page de manuel **man hier**. Parmi les répertoires à connaître :

- **/bin, /usr/bin** : exécutables principaux ;
- **/sbin, /usr/sbin** : exécutables pour l'administration ;
- **/usr/local** : fichiers associés aux programmes installés manuellement ;
- **/etc** : fichiers de configuration et scripts de démarrage ;
- **/root** : répertoire personnel de l'administrateur ;
- **/home** : répertoires personnels des utilisateurs ;

- **/var** : données dont la taille est variable ;
- **/var/spool/mail** ou **/var/mail** : boîtes de messagerie ;
- **/var/log** : journaux du système ;
- **/usr/include** : fichiers .h pour la programmation en C ;
- **/lib, /usr/lib** : bibliothèques pour la programmation en C ;
- **/dev** : fichiers spéciaux (périphériques) ;
- **/proc** : accès banalisé aux paramètres du noyau ;
- **/sys** (Linux 2.6+) : accès banalisé aux paramètres de certains périphériques ;
- **/tmp** : répertoire partagé pour les fichiers temporaires ;
- /usr/X11R6 : systèmes graphiques de fenêtrage [4].

Chaque répertoire possède deux sous-répertoires particuliers :

- « **.** » : représente le répertoire lui-même ;
- « .. » : représente le répertoire parent.

Enfin, sous UNIX, les fichiers dont le nom commence par « . » sont considérés comme cachés. Ainsi, le * du shell désigne tous les fichiers dont le nom ne commence pas par « . » et « .* » représente tous les fichiers cachés.

Remarque : la racine d'un système de fichiers a pour propriété « .. » = « . » . Cependant, lorsqu'un FS est monté ailleurs qu'en tant que « / » , sa racine récupère dynamiquement un nouveau père dans le FS d'accueil.

/usr est en principe conçu de telle sorte qu'il puisse être partagé par plusieurs machines via des montages réseau en lecture seule. En particulier, les utilitaires de **/bin** et **/sbin** doivent suffire à effectuer une maintenance locale du système en cas d'incident. Il existe un document décrivant en détails le contenu de tous ces répertoires. Il s'agit du *Filesystem Hierarchy Standard*. Il décrit les programmes qui doivent être présents dans **/bin** (**cat**, **cp**, **mount**, **ls**, **sh**, **su**, etc...).

3. Manipulations sur les fichiers UNIX

Quelques commandes à onnaître... Le résultat dépend de la validité des chemins (absolus ou relatifs au répertoire de travail courant) et des permissions sur les systèmes de fichiers (montage en lecture seule ou en lecture/écriture) et sur les éléments de ce système.

3.1. Fichiers normaux

- **cp [-p]** <**src**> <**dst**> : copier un fichier (-p : préserver les attributs) ;
- **mv** <**src**> <**dst**> : déplacer un fichier. En réalité, si on ne change pas de partition, **mv** ne fait que changer le répertoire qui référence l'inode du fichier.
- **rm [-i | -f]** <**dst**> : supprimer un fichier (-i : demander confirmation ; -f : forcer). En réalité, ne fait que décrocher l'inode du répertoire (pas d'effacement du contenu, ni même libération de l'inode si le compteur de liens reste strictement positif).

4. Cette séparation entre les fichiers liés à la gestion du mode graphique et les autres fichiers devient obsolescente.

- **shred <dst>** : surcharger plusieurs fois le contenu d'un fichier en vue de le détruire (Linux). Attention, les modes de gestion du système de fichiers sous-jacent et du disque par le système et par le contrôleur matériel ne permettent pas de garantir l'absence de rémanence de données sur le support physique à l'issue de l'opération.
- **ln <src> <dst>** : créer un lien durc. Ne fonctionne qu'en interne à une partition UNIX.
- **ln -s <src> <dst>** : créer un lien symbolique. Le lien doit être stocké sur une partition UNIX, et la source peut ne pas exister.
- **cat <src>** : afficher le contenu d'un fichier ;
- **stat <fichier>** : obtenir des informations détaillées sur un fichier ;
- **file <fichier>** : déterminer le type d'un fichier (à partir de son contenu, pas de son nom) ;
- **diff <fichier1> <fichier2>** : comparer ligne à ligne et affihe les différences entre deux fichiers ;
- **split -b<n> <fichier> <prefix>** : découper un fichier en fichiers de taille au plus <n> (le nom des fichiers créés commencera par <prefix>). Le fichier original peut en principe être reconstitué par : **cat <prefix>* > <fichier>**.

3.2. Répertoires

- **cd <dst>** : changer le répertoire de travail courant (*current working directory* ou CWD) (racine des chemins relatifs) ;
- **pwd** : afficher le répertoire de travail courant ;
- **mkdir <dst>** : créer un sous-répertoire ;
- **rmdir <dst>** : supprimer un sous-répertoire vide ;
- **cp -r [-p] <src> <dst>** : copier récursivement un répertoire ;
- **mv <src> <dst>** : déplacer un répertoire (la récursivité est implicite)[5] ;
- **rm -r [-f] <dst>** : supprimer récursivement un répertoire ;
- **ln -s <src> <dst>** : créer un lien symbolique ;
- **ls [-a] [-l] [-i] [-d] [-h] [<rep>]** : lister le contenu d'un répertoire ;
  -a (all) : montrer les fichiers cachés
  -l (long) : afficher les détails
  -i (inode) : lister les numéros d'inodes
  -d (directory) : voir le répertoire plutôt que son contenu
  -h (human-readable) : permet de choisir l'unité la plus pertinente pour afficher la taille des fichiers (non POSIX)
- **stat <rep>** : obtenir des informations détaillées sur un répertoire ;
- **du -k/-m/-h [<rep>]** : afficher l'espace occupé par un répertoire et ses sous-répertoires (fichiers compris) en Ko/Mo (-h fonctionne comme pour ls) ;
- **diff -r <rep1> <rep2>** : comparer récursivement les fichiers de deux répertoires et afficher les différences ligne à ligne.

5. Cette opération est rapide si on ne change pas de partition, puisque cela consiste uniquement à déplacer un point d'ancrage.

Remarque : à la création d'un sous-répertoire, deux liens sont immédiatement recensés (celui de son père vers lui, et son propre « . »).

Remarque : L'existence de liens durs pour les répertoires dépend de l'OS. Généralement, le système refuse la création de tels liens, qui peuvent rapidement conduire à une corruption du système de fichiers (création de boucles dans l'arborescence, risque de création d'orphelins lors du « détachement » de répertoires non vides ou d'incohérence dans la définition du « .. » , etc...). Sous Linux, **ln** prévoit une option -d pour le permettre, mais le noyau le refuse. Les liens symboliques vers les répertoires ne posent pas les mêmes problèmes (il est possible de les supprimer sans vider le répertoire, de ne pas les suivre dans un parcours récursif de l'arborescence[6]. La possibilité de créer des liens durs a été rétablie dans le noyau *MacOS X* pour le fonctionnement efficace de *Time Machine*. Enfin, il est à noter que contrairement à la copie, le lien (dur ou symbolique) résout le problème de la propagation des mises à jour et permet une économie d'espace disque.

4 Attributs de fichiers

4.1. Permissions

Chaque fichier du système VFS possède des attributs, dont les permissions. Ces permissions sont soit lues sur le système de fichiers auquel le fichier appartient (ex : EXT4), soit retraduites en fonction des options de montage (ex : FAT32). Les droits du superutilisateur (root) sur les fichiers ne sont pas limités par les permissions[7].

4.1.1. Permissions simples

Il existe, relativement à chaque fichier, trois types d'utilisateurs :

- le propriétaire du fichier ('u', poids 100[8]) ;
- les utilisateurs membres du groupe d'appartenance du fichier, moins le propriétaire ('g', poids 010) ;
- les autres utilisateurs ('o', poids 001).

Nota : 'a' représente 'ugo' (tous les utilisateurs).

Les permissions simples sont définies séparément pour chaque type d'utilisateur, et sont :

- la permission en lecture ('r', poids*4) : pour lire le contenu d'un fichier (**cat**) ou d'un répertoire (**ls**) ;
- la permission en écriture ('w', poids*2) : pour modifier le contenu d'un fichier (**echo** >) ou créer/supprimer/renommer des fichiers dans un répertoire (**touch**, **rm**, **mv**) ;
- la permission en exécution ('x', poids*1) : pour exécuter un fichier binaire ou un script (**./fichier**) ou entrer dans un répertoire (**cd**).

6. Par défaut, la commande **find** ne suit pas les liens symboliques. **find -L** les suit, mais utilise un algorithme pour éviter les boucles.

7. Il s'agit ici d'un raccourci. En effet, dans certains systèmes UNIX, les droits de root peuvent être éclatés en capacités POSIX. Certaines de ces capacités concernent le système de fichiers (CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_DAC_READ_SEARCH, CAP_FOWNER, CAP_FSETID, CAP_LINUX_IMMUTABLE, CAP_SETFCAP).

8. Attention, toutes les valeurs données pour les permissions sont exprimées en notation octale (base 8).

Les permissions peuvent s'exprimer en notation octale à partir des poids mentionnés plus haut. Ainsi, la lecture pour tous et l'écriture pour le propriétaire seulement s'écrit 644 (cf. **man chmod**). La commande **ls -l** présente les permissions sous la forme utilisateur / groupe / autres, un tiret (-) indiquant l'absence d'une permission. Ex : **rwxr-xr-x** représente la lisibilité et l'exécutabilité pour tous, et l'accès en écriture pour le propriétaire seulement (soit 755 en notation octale). Enfin, les propriétés des nouveaux fichiers sont déterminées par les règles suivantes :

- les permissions dépendent des options de création mais sont restreintes par le umask courant (cf. page de manuel) ;
- l'utilisateur propriétaire du fichier est son créateur (déterminé par l'uid effectif [9] courant) ;
- le groupe d'appartenance du fichier est soit le groupe d'appartenance du répertoire parent (convention BSD), soit le gid effectif de son créateur (convention System V) [10] ;
- voir plus bas le cas des bits s sur le répertoire parent.

La commande **id** permet d'affiher l'identité (uid) et le groupe principal (gid) de l'utilisateur courant, ainsi que la liste des groupes supplémentaires auxquels il appartient. L'identité et le groupe effectifs (cf. bits s, plus bas) sont également affichés s'ils sont différents.

Remarque : Les permissions ainsi accordées peuvent être globalement inhibées par les options de montage du système de fichiers sous-jacent. Ainsi, l'option **ro** (lecture seule) empêche toute modification du système de fichiers (création de fichiers, modification de données, changements de permissions, mise à jour de la date de dernier accès, etc...) et l'option **noexec** annule les permissions en exécution [11] apportées par les bits x.

4.1.2. Permissions étendues

Trois bits supplémentaires sont utilisés pour stocker les permissions dites étendues :

- bit setuid (premier bit s, poids 4000) :

  pour un fichier binaire exécutable (pas un script [12]), le programme tournera avec les privilèges effectifs (euid) du propriétaire du fichier plutôt que ceux de l'utilisateur lançant le programme (exemple : la commande de changement de mots de passe **passwd** est setuid root pour permettre l'accès en écriture au fichier des mots de passe)

  (rare) selon les UNIX et les options de montage, pour un répertoire accessible en écriture, les fichiers créés pourront appartenir au propriétaire du répertoire plutôt qu'à leur créateur (moyennant certaines restrictions)

- bit setgid (second bit s, poids 2000) :

  pour un fichier binaire exécutable (pas un script [13]), le programme tournera avec le groupe effectif (egid) correspondant à celui du fichier plutôt qu'à celui de l'utilisateur lançant le programme

9. Ces notions seront traitées plus en détails dans le cours sur les processus.
10. Linux adopte la convention System V sauf pour une partition explicitement montée avec l'option bsdgroups. Par ailleurs, ce système définit également des fsuid/fsgid qui peuvent surcharger les uid/gid effectifs pour la gestion des accès au système de fichiers.
11. Rappel : La permission en exécution n'est pas nécessaire pour sourcer un script.
12. Certains systèmes, comme OpenBSD, traitent les bits 's' des scripts comme ceux des exécutables binaires. Cette possibilité est dangereuse et ne doit pas être utilisée.
13. Sauf sur certains systèmes, cf. plus haut.

en convention System V, pour un répertoire accessible en écriture, les fichiers créés seront associés au groupe du propriétaire du répertoire, plutôt qu'à celui de leur créateur

- sticky bit (bit t, poids 1000) :

  (rare, historique) sur certains UNIX, pour un fichier binaire exécutable, le mode de gestion du fichier en mémoire peut être affecté par ce bit (le détail dépend de l'OS : persistance dans la zone de swap, chargement sans cache, etc...)

  pour un répertoire accessible en écriture, les fichiers créés ne seront supprimables que par leur propriétaire (ou celui du répertoire). Le répertoire /tmp fonctionne selon ce principe

Avec **ls -l**, les permissions étendues sont affichées par dessus les permissions en exécution. Une majuscule ('S' au lieu de 's') indique que la permission en exécution affichable au même endroit est absente.

Remarque importante : L'ajout d'un bit 's' à un exécutable doit correspondre à une **délégation maîtrisée de privilèges** (fourniture d'une fonctionnalité de haut niveau légitime nécessitant, à bas niveau, l'exploitation de privilèges potentiellement dangereux). Cette délégation relève, au niveau applicatif, de la même logique que celle appliquée par l'OS (noyau) lorsqu'il « délègue » son privilège d'accès physique au disque sous forme de fonctions d'accès aux fichiers, imposant notamment au passage le contrôle des permissions.

Nota : l'option de montage **nosuid** permet d'ignorer les éventuels bits 's' des exécutables du système de fichiers concerné. Il s'agit d'une précaution à prendre lors du montage d'un support moins maîtrisé que le système local (clé USB, serveur de fichiers distant).

4.2. Changement des attributs

- **chmod [-R]** <**nnnn**> <**dst**> : changer les permissions[14] (en octal) (-R : récursif). *Nota bene* : Autres formats de spécification des permissions :

  a+x : ajouter l'exécution pour tous

  o-rwx : enlever toutes les permissions hors propriétaire et groupe

  u+s : mettre le premier bit s

  etc...
- **chown [-R]** <**user**>**[:**<**group**>**]** <**dst**> : changer le propriétaire et le groupe d'un fichier[15] (-R : récursif).
- **chgrp [-R]** <**group**> <**dst**> : changer le groupe d'un fichier (-R : récursif) ;
- **touch** <**dst**> : mettre à jour la date de dernière modification d'un fichier, en le créant vide si nécessaire.

5. Gestion des accès aux fichiers

5.1. Structure d'accès aux fichiers

L'OS (noyau) conserve, pour chaque processus, une table de fichiers[16] ouverts. Les entrées de cette table pointent vers des structures internes du noyau, qui décrivent les propriétés des accès à un

14. Seul root peut rajouter un bit **setgid** sans être membre du groupe associé au fichier.
15. Selon les OS, le fait de donner un fichier à un autre utilisateur ou à un groupe dont on n'est pas membre est une opération privilégiée (réservée à root) ou non. Les éventuels bits s sont généralement effacés.
16. au sens large : fichiers et répertoires de VFS, *tubes*, *sockets*, etc... (cf. cours suivants).

fichier, et notamment :

- un compteur de références sur la structure ;
- le mode d'accès obtenu sur le fichier (lecture, écriture) ;
- la position courante dans le fichier ;
- une référence vers une autre structure interne du noyau, qui décrit l'inode concerné.

Une telle structure est générée à chaque opération d'ouverture de fichier (y compris si le fichier est déjà ouvert par ailleurs)[17], avec un compteur de références initialement à 1. Ce compteur est incrémenté de façon transitoire lors de certaines opérations sur le fichier, ou durablement par des opérations de type dup/dup2 (cf. cours n°3) ou fork (cf. cours n°4), qui conduisent à un partage de la structure entre plusieurs descripteurs de fichiers. Il est décrémenté durablement lors de la fermeture d'un descripteur de fichier qui lui est associé. La structure est détruite lorsque le ompteur tombe à zéro.

Remarque fondamentale : Le contrôle des permissions ne s'effectue qu'à l'ouverture d'un fichier, c'est-à-dire lors de la création d'une nouvelle structure de gestion des accès. Les accès ultérieurs au fichier sont ensuite comparés au mode (lecture seule, etc...) selon lequel il a été ouvert.

Un exemple de telle structure est donnée par les figures 1 et 2. Ces figures mettent en scène deux processus qui accèdent au même fichier, mais à travers des structures d'accès distinctes.

Les trois premières entrées de la table ont un sens particulier pour la couche utilisateur : 0 représente l'entrée standard du processus (source de **getchar**), 1 la sortie standard (destination de **printf**) et 2 la sortie d'erreur (destination de **perror**). Ces trois descripteurs se transmettent généralement lors de la création de nouveaux processus.

Pour des raisons d'optimisation, les accès en écriture ne sont pas répercutés instantanément sur le disque par l'OS (dépend des options de montage). Elles sont par contre prises en compte au niveau des caches VFS pour assurer la cohérence en cas de relecture après une écriture, y compris si le programme qui lit n'est pas celui qui a écrit. La commande **sync** permet de demander explicitement à l'OS d'effectuer toutes les opérations d'écriture en attente (sans réelle garantie de résultat). Au plus tard, la synchronisation a lieu lors du démontage de la partition concernée.

Remarque : On prendra garde à ne pas confondre les différents niveaux de caches intervenant lors d'une opération d'écriture :

- le(s) cache(s) applicatif(s), propre(s) à chaque programme, qui résulte(nt) de la gestion explicite de *buffers* et/ou de l'utilisation des primitives C de haut niveau (cf. section 6.1) ;
- le cache VFS (descripteurs d'inodes, blocs de données), géré par l'OS, qui maintient la cohérence entre appels à **write** et à **read**, y compris par des programmes différents ;
- le cache physique du contrôleur disque, qui peut retarder l'écriture effective sur le disque lorsque l'OS lui transmet les mises à jour.

17. La structure noyau décrivant un inode ouvert est quant à elle unique au niveau de l'OS quel que soit le nombre de fois où le fichier a été ouvert (et les modes d'ouverture associés). Elle sert en particulier à stocker les informations relatives au cache VFS (voir plus bas) et aux verrous sur les fichiers (cf. cours n°3).

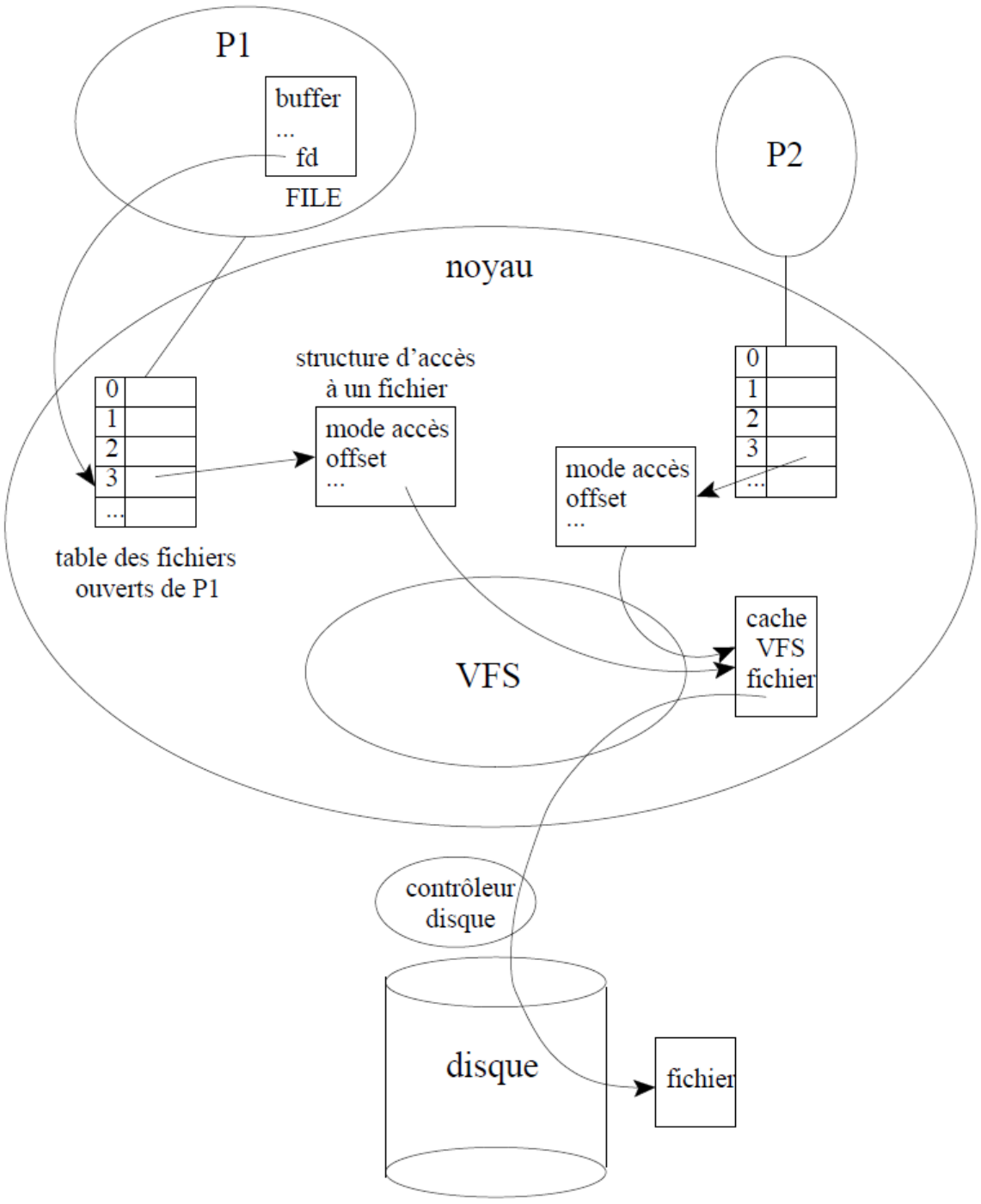

FIGURE 1 – Ouverture d'un même fichier par deux processus.

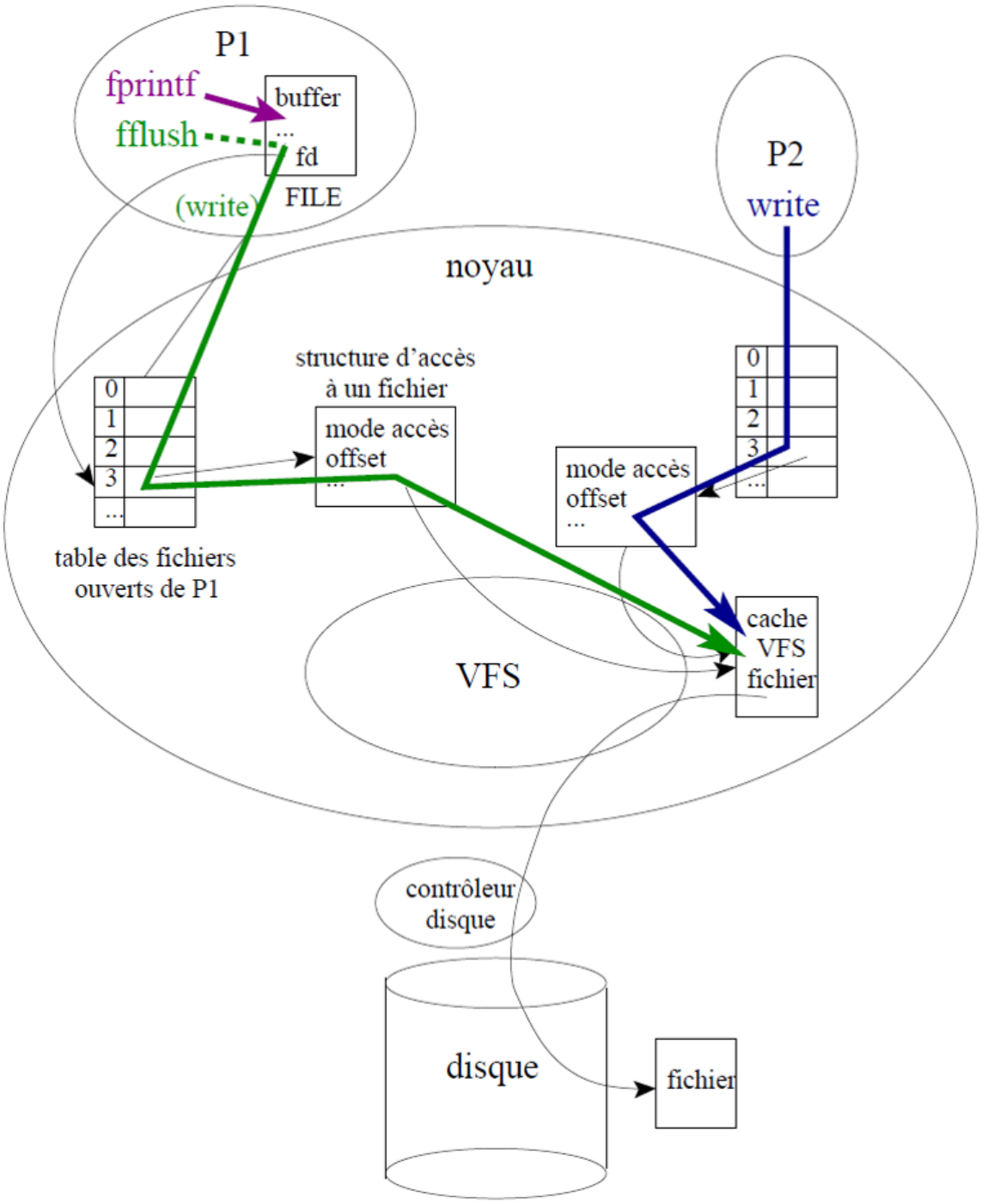

FIGURE 2 – Détails des opérations d'écriture dans un même fichier par deux processus.

5.2. Inodes

Au sein du VFS, les fichiers (ou répertoires) sont représentés sous la forme d'inodes. Les inodes sont composés de méta-données relatives au fichier, notamment :

- l'identifiant (uid) du propriétaire ;
- l'identifiant (gid) du groupe d'appartenance ;

- des attributs normaux (type de fichier et permissions) ;
- des attributs spéciaux (flags : fichier non modifiable, etc...), rarement utilisés (cf. **man chattr** et **man lsattr**) sous Linux (les équivalents BSD sont **chflags** et **ls -lo**) ;
- les éléments nécessaires au support des listes de contrôle d'accès (ACL, non gérées par défaut sous Linux) et autres attributs étendus ;
- un compteur de liens (nombre de fois où cet inode est référencée). L'inode n'est libérée que lorsque ce compteur passe à 0.
- les dates de dernier changement d'attribut (**time**[18]), de dernière modification de contenu (**mtime**), de dernier accès (**atime**) et de suppression (**dtime**) ;
- une taille (en octets).

Remarque : Bien que peu utilisé, deux attributs étendus peuvent se révéler très utiles : l'attribut **immutable**, qui offre des garanties fortes d'intégrité sur un fichier (il ne peut être modifié ni supprimé) et l'attribut **append only** qui garantit qu'un fichier ne pourra être ouvert qu'en mode ajout. Seul root (ou sous Linux, un utilisateur disposant de la capacité CAP_LINUX_IMMUTABLE) peut altérer es deux attributs.

Il est intéressant de noter que le nom par lequel on accède à un fichier n'est pas défini au niveau de son inode mais à celui du répertoire auquel il est lié.

Le système de fichiers EXTx utilise une structure similaire pour gérer les méta-données sur le support physique. Cependant, pour d'autres systèmes de fichiers comme FAT, les méta-données sont plus pauvres, et c'est le noyau qui remplit les informations manquantes au sein des inodes du VFS.

6. APIs pour le langage C

Il existe deux APIs C pour l'accès aux fichiers standard.

6.1. API de haut niveau (ANSI)

Les descripteurs de fichiers de haut niveau sont des **FILE *** (fichiers standard : *stdin*, *stdout* et *stderr*). Les accès utilisant cette API sont bufferisés au niveau du programme qui les utilise (cache applicatif). Les primitives C sont :

- **FILE *f = fopen(const char *name, "r" / "r+" / "w" / "a")** : ouverture (chemin absolu ou relatif) ;
- **int ret = fclose(FILE *f)** : fermeture ;
- **int ret = fprintf(FILE *f, const char *format, args...)** : écriture ;
- **fwrite, fputc, fputs** : autres fonctions d'écriture (enregistrements, caractères, lignes) ;
- **int ret = fscanf(FILE *f, const char *format, &args...)** : lecture (À ÉVITER ! utiliser plutôt **fgets**) ;
- **fread, fgetc, fgets** : autres fonctions de lecture (enregistrements, caractères, lignes) ;
- **int ret = fseek(FILE *f, long offset, SEEK_SET / SEEK_CUR / SEEK_END)** : déplacement (absolu ou relatif) ;
- **int ret = feof(FILE *f)** : teste si on a atteint la fin d'un fichier ;

18. parfois appelée, à tort, date de création.

- **int ret = fflush(FILE *f)** : synchroniser les écritures au niveau du programme (vider le buffer en cours). **fflush(NULL)** permet une synchronisation de tous les FILE *.
- **setbuf, setvbuf** : modifier la bufferisation d'un FILE *.

Remarque : **printf(...)** n'est qu'un raccourci pour **fprintf(stdout,...)**. Attention aux prototypes des fonctions. Ex : **fgetc** renvoie un int, et non un char. Ceci permet par exemple de signaler qu'on est à la fin du fichier (renvoie EOF, qui n'est pas codable sur un char).

Remarque : L'API de haut niveau utilise en sous-main l'API de bas niveau décrite plus bas, avec les retards et regroupements liés à la bufferisation. Attention, **fflush** ne réalise PAS de synchronisation au niveau OS (**fsync**).

6.2. API de bas niveau (POSIX)

Cette API est au contact direct des fonctionnalités offertes par l'OS. Les fichiers sont représentés par des entiers qui correspondent aux descripteurs de fichiers ouverts du processus tels que gérés par l'OS. Les descripteurs de fichiers standard sont STDIN_FILENO (0), STDOUT_FILENO (1), STDERR_FILENO (2). Les écritures formatées **fprintf** sont à remplacer par un préformatage via **snprintf** (**printf** dans une chaîne de caractères).

- **int fd = open(const char *name, O_RDONLY/O_WRONLY/O_RDWR [| O_TRUNC] [| O_APPEND] [| ...] [| O_CREAT [| O_EXCL], mode_t mode])** : ouverture. Il faut préciser le mode potentiel de création si on utilise l'option O_CREAT [19]. Ce mode sera implicitement restreint en fonction du **umask** courant. Il s'écrit sous la forme : S_IWUSR | S_IXGRP | S_IROTH | ...
- **int ret = close(int fd)** : fermeture ;
- **ssize_t rlen = read(int fd, void *buf, size_t len)** : lecture ;
- **ssize_t wlen = write(int fd, const void *buf, size_t len)** : écriture ;
- **off_t new_off = lseek(int fd, off_t offset, SEEK_SET/SEEK_CUR/SEEK_END)** : déplacement (absolu ou relatif) ;
- **int ret = fsync (int fd)** : synchroniser les écritures au niveau de l'OS (données et méta-données du cache VFS).

Sur les systèmes Linux 32 bits, le curseur de position, de type off_t, est un entier signé sur 32 bits. Il est possible de manipuler des fichiers de plus de 2 Go en ajoutant l'option O_LARGEFILE à open et en utilisant des variantes 64 bits (off64_t) des fonctions travaillant sur les off_t (ex : lseek64 [20]).

Notion importante : Une **race condition** est un type de vulnérabilité mettant en jeu une sucession d'opérations réalisées sur une même ressource par un programme légitime en supposant implicitement que la ressource n'est pas modifié par un tiers entre temps, sans que cette hypothèse soit réalisée. Par exemple, si on écrit dans un script :

```
touch fichier
echo toto >> fichier
```

on n'a aucune garantie que le fichier n'a pas été modifié entre les deux instructions. On dit aussi qu'il n'y a pas atomicité de l'ensemble des deux commandes : il s'agit de deux commandes distinctes. Ce type de vulnérabilités est difficile à repérer, mais a un impact bien réel.

19. Il est possible de créer le fichier avec des permissions inférieures à celles requises pour l'ouvrir avec le mode d'accès demandé.
20. voir aussi _llseek.

6.3. Passage d'une API à l'autre

Deux fonctions permettent de passer d'une API à l'autre :

- **FILE *f = fdopen(int fd, "r"/"r+"/"w"/"a")** : génère un FILE * à partir d'un descripteur simple (dans un mode d'accès compatible) ;
- **int fd = fileno(FILE *f)** : récupère le descripteur bas niveau associé à f.

De façon générale, essayer de travailler avec l'API bas niveau (sauf pour les sorties standard et d'erreur) et s'y tenir !

Ce document est inspiré du « Cours Système n°2 : les Fichiers (1/2) » délivré par monsieur Olivier Levillain dans les cours dispensés à l'Agence nationale de la sécurité des systèmes d'information (Octobre 2012).