

Systèmes de Gestion de Fichiers

SYR1-L3 Info & Miage

1

Partie I : Organisation des fichiers

Contexte et rappels
Définitions
Protection des fichiers

SYR1-L3 Info & Miage

2

Plan

1. Généralités
 - a. Qu'est ce qu'un SGF ?
2. Structures de données utilisées dans un SGF
 - a. Données et méta données
 - b. La notion d'Unité d'Allocation
 - c. Les Tables d'Implantations
 - d. La Table des U.A libres/occupés
 - e. Catalogue(s) et hiérarchie de fichiers
3. Protection des fichiers dans un SGF
 1. Pourquoi a-t-on besoin de mécanismes de protection ?
 2. Notion de *liste de contrôle d'accès*
 3. *Matrice d'accès* vs *liste d'accès*

SYR1-L3 Info & Miage

3

Plan

4. Organisation des fichiers sous MS-FAT
 - a. Présentation des SGF de type *FAT*
 - b. La *File Allocation Table*
 - c. La gestion des catalogues
4. Organisation des fichiers sous Unix
 - a. Notion de *inode* et MEO des catalogues sous Unix
 - b. Organisation des volumes Unix
 - c. Protection des fichiers sous Unix
4. Comparaison Unix/FAT

SYR1-L3 Info & Miage

4

I . 1 : Généralités

SYR1-L3 Info & Miage

5

1. Introduction : à quoi sert le SGF ?

- Le SGF est un composant du S.E., son rôle est de :
 - Permettre de *désigner* des fichiers par un *nom* : retrouver des fichiers parmi tous les fichiers connus (on parle de *désignation*)
 - Offrir une *protection* des fichiers : sécurité, confidentialité des informations.
- La mise en œuvre dépend du type de système
 - Sur un appareil photo numérique, le S.E. utilise un SGF pour lire/écrire les photos sur la carte FLASH
 - Un seul utilisateur, pas besoin de protection
 - À l'Université, les données des utilisateurs sont stockées sur des serveurs disposant de plusieurs dizaines de disques.
 - Multi-utilisateurs, protection des fichiers importante, haute disponibilité, tolérance aux fautes, sauvegarde de sécurité, etc.

SYR1-L3 Info & Miage

6

1. Quelques systèmes de fichiers

- Les système de fichiers utilisant le principe d' *inode* (Unix)
 - Linux utilise ce type de SGF (ext2fs, ext3fs)
- Les systèmes de fichiers de type FAT
 - Utilisé par le système MS-DOS dans les années 80
 - À évolué afin de suivre la capacité de stockage des disques
 - FAT12, FAT16, puis FAT32
 - Les formats FAT16 et FAT32 sont toujours utilisés pour les stockages amovibles (clés USB, DD externes, etc.)
- Remarque : Windows 7 & XP utilisent NTFS
 - Spécifications et fonctionnement de NTFS non disponibles
 - Complètement différent des SGF de type FAT

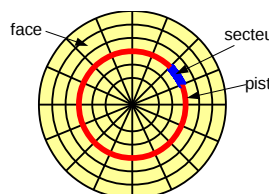
Dans ce cours on va étudier l'approche basé sur les *inode* (UNIX) et l'approche FAT (Windows)

SYR1-L3 Info & Miage

7

1. Rappels – Le matériel

- Organisation interne d'un disque dur



- ◆ L'élément adressable d'un disque est le *secteur* (512 octets).
- ◆ Un secteur est repéré par son adresse physique ($n^{\circ} \text{secteur}$)

- Aujourd'hui il existe d'autres supports de stockage
 - DVD-ROM, CD-ROM, disquette, etc : similaire au disque
 - Stockage FLASH : fonctionnement parfois très différent (mais on ne l'abordera pas ici)

SYR1-L3 Info & Miage

8

1. Rappels – E/S de base

- **Secteur** = plus petit élément adressable sur un disque
 - Un secteur occupe 512 octets.
- **Bloc d'E/S** = unité d'échange des E/S physiques
 - nombre entier (en général 2ⁿ) de **secteurs contigus** sur le disque.
- Les E/S physiques nous fournissent deux fonctions
 - `int readblock(int addr_sector, char* buffer)`
 - `int writeblock(int addr_sector, char* buffer)`
- Les E/S de base fonctionnent de manière *synchrone*
 - Elles sont lentes (de l'ordre de la milliseconde)
 - *Un processeur moderne exécute une instruction en ~1ns*
 - Il faut donc essayer de minimiser le nombre d'appels à ces fonctions

SYR1-L3 Info & Miage

9

1. SGF : rôle et fonctionnement

- Gestion de l'espace disque
 - Le SGF doit savoir à tout moment quels sont les secteurs *libres* ou *occupés* sur le disque.
- Liaison/désignation:
 - Le SGF doit pouvoir retrouver les données d'un fichier sur le disque à partir de son *nom externe*.
- Rémanence
 - On doit pouvoir arrêter la machine, puis la redémarrer sans que cela ait des conséquences sur le SGF (stockage permanent)
- Principe : on stocke deux types d'infos sur le disque
 - Les *données* des fichiers proprement dits
 - Des *méta données* qui décrivent l'état du SGF

SYR1-L3 Info & Miage

10

I.2 : Structures de données utilisées dans un SGF

SYR1-L3 Info & Miage

11

2. Définition : unités d'allocation

- On découpe le disque en **Unités d'Allocation** (*cluster*)
- **Unité d'Allocation** : plus petite quantité d'espace allouable
 - Nombre entier (en général 2ⁿ) de secteurs consécutifs
 - La taille des U.A. est un multiple de la taille des blocs d'E/S
 - Les fichiers sont découpés en une collection d'U.A.
 - Une même U.A. ne peut pas être partagée entre deux fichiers
- Le choix de la taille des U.A. a des conséquences :
 - L'utilisation d'U.A. de grande taille = gaspillage de place pour les petits fichiers (qui occupent au minimum une U.A. complète).
 - Sur les PC les fichiers sont de + en + gros (*mp3*, *divx*, etc.) la tendance va donc vers une augmentation de la taille des U.A.
 - Remarque : la taille des U.A. a d'autres incidences que nous allons étudier dans la suite.

SYR1-L3 Info & Miage

12

2. Définition : données et méta données

- Les données des fichiers du SGF
 - On découpe les données des fichiers en *unités d'allocation*
 - Les données d'un fichier seront réparties sur une ou plusieurs *unités d'allocation*.
- Les *méta données* qui correspondent à l'état du SGF
 - On associe à chaque fichier une *table d'implantation*, qui contient la liste des *unités d'allocation* du fichier
 - On associe à chaque fichier un *descripteur de fichier* qui regroupe les informations le concernant.
 - On range la liste des fichiers du SGF dans un (ou plusieurs) *catalogue(s)*.
 - On conserve la liste des secteurs libres/occupés sur le disque.

SYR1-L3 Info & Miage

13

2. Définition : Table d'Implantation

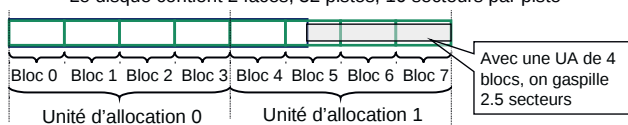
- **Table d'implantation** : liste des U.A. associées à un fichier
 - Dans la suite on utilisera l'abréviation *T.Impl.*
 - Cette table doit être stockée sur le disque (info permanente)
 - Lors des accès à ce fichier, on utilise les informations de cette table pour retrouver les @ des U.A. contenant les données du fichier
 - Elle contient les *adresses disque* des U.A. formant le fichier
 - *nb. U.A.* : nombre d'U.A. utilisées par le fichier
 - *[@U.A.]* : nombre de bits nécessaires au codage d'une adresse d'U.A.
- La taille de cette table dépend de plusieurs paramètres :
 - la taille du fichier à stocker sur le disque (nb. U.A. du fichier),
 - la taille des unités d'allocation (*#U.A. du fichier*, *[@U.A.]*),
 - et le nombre d'U.A. disponibles sur le disque (*#U.A.*)

SYR1-L3 Info & Miage

14

2. Table d'implantation : exemple

- Exemple
 - Un fichier de 2700 octets, un secteur = 512 octets
 - Un bloc d'E/S = un secteur, une U.A. regroupe 4 blocs
 - Le disque contient 2 faces, 32 pistes, 16 secteurs par piste



- Quelle est la taille de la table d'implantation ?
 - Le fichier est formé de deux U.A. Il faut y stocker deux adresses
 - Le disque contient au plus $N_{U.A.} = N_{face} * N_{piste} * N_{sect/piste} / N_{sect/UA}$ U.A.
 - Une @U.A. peut donc être codée avec $\log_2(N_{U.A.})$ bits (arrondi à la valeur supérieure)
 - Taille $T_{Impl} = \#U.A. * |@U.A.| = 2 * (\log_2(256)) = 16 \text{ bits} = 2 \text{ octets}$

SYR1-L3 Info & Miage

15

2. Définition : Descripteur de fichier

- Utilisé pour stocker des *informations* sur un fichier
 - Chaque fichier du disque dispose de son propre descripteur
 - Un descripteur contient des *informations permanentes*
 - Il doit donc être stocké sur le disque
- Quelles informations stocke-t-on dans ce descripteur ?
 - **nom** : nom externe du fichier (unique à un fichier)
 - Parfois le *nom externe* n'est pas stocké dans le descripteur (Unix)
 - **type** : rôle du fichier dans le système (cf. suite du cours).
 - **taille** : nombre d'octets utilisés par le fichier.
 - **propriétaire** : identifiant de l'utilisateur ayant créé le fichier.
 - **protection** : droits d'utilisation du fichier.
 - **table d'implantation** : liste des adresses disque des U.A.
 - **divers** : date de création, date de dernière modification.
 - etc.

SYR1-L3 Info & Miage

16

2. Définition: Table des blocs libres/occupés

istic

- Il faut connaître les secteurs libres/occupés pour :
 - Allouer de l'espace à un fichier (création, extension)
 - Libérer de l'espace (suppression de fichiers)
- On pourrait parcourir les *T.Impl.* de tous les fichiers
 - On connaîtrait ainsi l'ensemble des U.A. utilisées par tous les fichiers présents sur le SGF.
 - Ces tables d'implantation sont parfois dispersées sur le disque
 - Accès très lent à cause des déplacements du bras de lecture.
- On conserve plutôt sur le disque une *image* de cette info.
 - Par exemple : une liste des U.A. libres sur le disque.
 - Facilite la gestion des U.A (accès à l'info + simple)
 - Information redondantes par rapport aux *T.Impl.*
 - Gaspillage de place (info redondante)
 - Risque d'incohérences en cas d'arrêt brutal du système

2. Désignation de fichiers : le catalogue

istic

- Attribuer un *nom/identifiant* à un objet permet
 - d'identifier cet objet, c.-à-d. de le distinguer des autres
 - d'accéder à cet objet, c.-à-d. de l'utiliser
- Dans le cas d'un objet de type *fichier* il s'agit
 - D'établir la correspondance (c.-à-d. faire la *liaison*) entre un *nom externe* et un *descripteur de fichier* du SGF.
 - On rappelle qu'un descripteur de fichier
 - Identifie (normalement) de manière unique un fichier
 - Permet d'accéder aux données du fichier grâce à la *T.Impl.*
- Cette opération de liaison se fait par *le(s) catalogue(s)*
 - Leur MEO peut varier d'un système à un autre
 - On va étudier deux MEO de SGF : *Unix* et *Microsoft FAT*

2. Définition : catalogues hiérarchisés

istic

- Comment structurer l'*espace de nommage* des fichiers
 - Espace de nommage* = ensemble des noms externes possibles
 - On crée des *environnements de désignation* différents pour ne voir à un moment donné qu'un sous-ensemble des noms de fichiers du SGF
- Ici, structuration=organisation des fichiers en *répertoires*
 - Un répertoire est en fait un fichier *catalogue*
 - Ce fichier *catalogue* permet de référencer d'autres fichiers ou d'autres répertoires
 - On dispose ainsi d'une hiérarchie de catalogues, avec un fichier *catalogue racine* (noté / sous Unix)
- Le SGF devra distinguer deux types de fichiers
 - Les *fichier ordinaires* (contiennent des données de l'utilisateur)
 - Les *répertoires* (contiennent des informations du SGF)

2. Avantages des catalogues hiérarchisés

istic

- Permet de structurer logiquement le SGF
 - On range les programmes dans un répertoire (*/bin*),
 - Les données d'un utilisateur dans un répertoire qui lui est propre (*/user/dupond*), etc.
- ```
graph TD; Root[" / "] --- Bin[" bin/ "]; Root --- User[" user/ "]; Root --- Dev[" dev/ "]; Bin --- ls; Bin --- rm; Bin --- cp; User --- tp2c[" tp2.c "];
```
- Facilite la protection des fichiers
    - On range tous les programmes systèmes dans un même répertoire, auquel on interdit l'accès en écriture aux utilisateurs.
  - Possibilité d'avoir plusieurs fichiers avec le même nom **simple**
    - Chaque utilisateur peut ainsi avoir son propre fichier **tp2.c**
    - Exemple : un fichier **tp2.c** dans le répertoire */user/dupond/* et dans le répertoire */user/durand/*

2. Catalogue hiérarchisé

istic

- Dans un système hiérarchisé, on distingue :
  - Le **nom simple** (nom figurant dans le répertoire)
  - Le **chemin d'accès** (*pathname*) qui est formé par la suite des noms de répertoires depuis la racine jusqu'à l'objet désigné.
  - Le **nom externe** du fichier est le **chemin d'accès + le nom simple**
- Exemple (sous UNIX)

nom simple

/share/L3info/SYS2/toto.txt

chemin d'accès
- Exemple (sous Microsoft Windows)

nom simple

G:\L3info\SYS2\toto.txt

chemin d'accès

2. Catalogue hiérarchisé

istic

- Il existe des mécanismes simplifiant la désignation
  - Évitent d'utiliser le *nom externe* complet à chaque fois
- Notion de *Working Directory* (répertoire courant)
  - nom externe = chemin d'accès du W.D. + nom simple
  - chemins d'accès relatifs (on note *..* / le répertoire père).
- Utilisation de règles de recherche
  - La variable **\$PATH** permet d'indiquer la liste des répertoires contenant les exécutables binaires.
  - sous Unix, la variable **\$LD\_LIBRARY\_PATH** permet d'indiquer la liste des répertoires contenant les bibliothèques dynamiques.

2. SGF Multi-Utilisateurs

istic

- Quels sont les problèmes propres à ce type de SGF ?
  - Les *conflits de nom* entre utilisateurs
    - Deux utilisateurs A et B souhaitent créer un fichier **TP1.asm**
  - Il y a un besoin de *partager* des fichiers
    - Un utilisateur A se sert des fichiers d'un utilisateur B
    - Tous les utilisateurs accèdent aux mêmes fichiers exécutables
  - Besoin de *protection* supplémentaire
    - Une erreur chez un utilisateur ne doit pas avoir de conséquence pour les autres utilisateurs
- Solutions mises en œuvre :
  - On introduit un nom d'utilisateur (+ mot de passe) et un répertoire maison (*home directory*) associé à chaque utilisateur
  - L'utilisateur A doit pouvoir désigner les fichiers de l'utilisateur B si celui-ci l'y autorise. Les répertoires personnels sont donc visibles et intégrés à la hiérarchie du SGF.

I . 3 : Protection des fichiers dans un SGF

### 3. Protection dans un SGF Multi-Utilisateurs

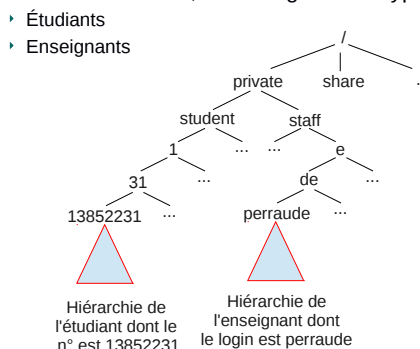
- Quels sont les objectifs de la protection
  - Assurer l'*intégrité* de l'information sur le SGF, celle-ci ne doit pas pouvoir être modifiée de manière incontrôlée (par erreur ou par malveillance)
  - Assurer la *confidentialité* de l'info (n'importe qui ne peut pas consulter n'importe quoi)
- Le problème est crucial dans un S.E. Multi-Utilisateurs
  - Dans un S.E. mono utilisateur (ex : appareil photo numérique) il faut juste se protéger contre ses propres erreurs.
- La protection est basée sur le principe de listes d'accès
  - À la base de tous les mécanismes : on a une identification de l'utilisateur lors de sa connexion (nom de login + mot de passe)

SYR1-L3 Info & Miage

25

### 3. Système Multi-Utilisateurs

- Sur le réseau ISTIC, on distingue deux types d'utilisateurs



SYR1-L3 Info & Miage

26

### 3. Notion de liste de contrôle d'accès

- On utilise la notion de *couple d'accès*
  - couple d'accès* = (objet, droits sur cet objet)
  - état des droits d'accès = ensemble des couples d'accès
  - À un instant donné (dans un contexte donné) le SGF est dans un certain *état des droits d'accès*
- L'*état des droits d'accès* du SGF peut évoluer
  - Parce que l'on a modifié les droits d'accès à un objet
    - Exemple : commande **chmod** sous Unix
  - Parce qu'on change d'environnement d'exécution
    - Exemple : changement d'utilisateur, ou de mode d'exécution
- Notion de domaine de protection
  - domaine de protection* = ensemble des couples d'accès définissant les objets accessibles à un instant donné

SYR1-L3 Info & Miage

27

### 3. Notion de liste de contrôle d'accès

- Les domaines de protection peuvent être organisés en
  - Matrice d'accès* : UNIX et assimilés (LINUX, FreeBSD, etc.)
  - Liste d'accès* : NTFS (Windows XP), Multics (ancêtre d'Unix)
- Remarque :
  - Les SGF de MAC OS X gèrent les deux (liste et matrice)
  - Sur les systèmes de type FAT (SGF de WIN95, MS-DOS) il n'existe pas de véritable mécanisme de protection

SYR1-L3 Info & Miage

28

### 3. Exemple de matrice d'accès

- Dans cet exemple
  - Domaines = utilisateurs (*dupond, durand, martin*)
  - Objets = fichiers (**toto.txt**, **tp2.java**, **prog.exe**)
- Principe de fonctionnement
  - On modifie les droits d'accès du fichier en fonction du domaine d'utilisation dans lequel on se trouve

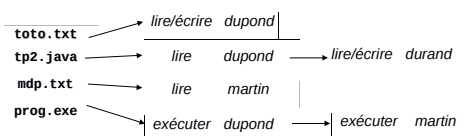
| objets<br>domaine | toto.txt    | tp2.java    | mdp.txt | prog.exe |
|-------------------|-------------|-------------|---------|----------|
| dupond            | Lire/écrire | Lire        |         |          |
| durand            |             | Lire/écrire |         | Exécuter |
| martin            |             |             |         | Exécuter |

SYR1-L3 Info & Miage

29

### 3. Liste de contrôle d'accès

- Quand la matrice est creuse, on la stocke par colonne
  - Pour chaque objet on a une *liste de contrôle d'accès*
  - Chaque élément de la liste contient la description des droits d'accès à cet objet pour un domaine particulier



SYR1-L3 Info & Miage

30

### 3. Liste d'accès

- Difficultés de mise en œuvre :
  - L'espace occupé par cette liste peut varier au cours du temps
  - On ne peut donc pas stocker cette liste dans un descripteur de fichier dont la taille est figée.
  - Cela complique le travail du SGF
- Solution possible :
  - On regroupe les usagers en *groupes*, un usager pouvant appartenir à plusieurs groupes.
  - Le domaine correspond alors à l'ensemble (*nom de groupe, nom d'utilisateur*)
  - Un usager particulier peut alors avoir des droits différents en fonction du groupe de connexion.

SYR1-L3 Info & Miage

31

### 3. Limites des listes de contrôle d'accès

- Ne protègent pas contre une attaque *physique*
  - Si on peut extraire le disque dur de la machine, on peut accéder au contenu du disque en court-circuitant le SGF.
  - Il est alors possible de retrouver (modifier) le contenu de fichiers auxquels on n'a normalement pas accès.
    - Parfois on peut même retrouver le contenu de fichiers effacés
- Évolution des mécanismes de protection
  - Pour assurer la *confidentialité des données*, l'utilisation de cryptographie est de + en + fréquente.
  - On **chiffre** les données des fichiers du disque à l'aide d'une *clé de chiffrement*.
  - Seul le propriétaire de la clé (c.-à-d. le propriétaire du fichier ou de la machine) peut **déchiffrer** (et donc lire) le contenu du fichier/disque.

SYR1-L3 Info & Miage

32

## I . 4 : Organisation des fichiers sur les SGF (Microsoft) FAT

SYR1-L3 Info &amp; Miage

33

## 4. Les SGF de type FAT

- FAT12 : premières versions de MS-DOS (1977)
  - Table à 2<sup>12</sup> entrées, avec des U.A. variant entre 1Kio et 4Kio
  - Supporte au maximum des partitions de 32Mio
  - Taille de la FAT ~ 100Kio,
  - Encore utilisé pour les disquettes sous Windows XP
- FAT16 : MS-DOS 4.0 (1988)
  - Table à 2<sup>16</sup> entrées, avec des UA variant entre 1Kio et 32Kio
  - Supporte au maximum des partitions de 4Gio
  - Taille de la FAT ~ 1Mio
  - Utilisé pour les disques USB
- FAT32 : Windows 95 (1996)
  - Table à 2<sup>28</sup> entrées, avec des UA variant entre 1Kio et 32Kio
  - Taille de la FAT ~ 80Mio, taille maximum d'une partition 8Tio (en théorie, 2Tio en pratique)

SYR1-L3 Info &amp; Miage

34

## 4. Table d'implantation et FAT

- Basée sur une table (*File Allocation Table*)
  - Le nombre d'entrées de la table dépend de la version
    - FAT16 = 2<sup>16</sup> entrées, FAT32=2<sup>28</sup> entrées, etc.
  - Une entrée de la FAT contient un numéro qui identifie à la fois
    - Une U.A du disque contenant les données (n° d'U.A)
    - Une entrée de la FAT contenant les info de l'U.A suivante
  - Permet de mettre en œuvre des listes chaînées d'@UA
- L'implantation des fichiers est stockée dans la FAT
  - Tous les fichiers utilisent la même table FAT
  - Le descripteur de fichier contient simplement une @Entrée FAT
    - Cette entrée contient le numéro de la 1ere U.A du fichier, et l'adresse de l'entrée contenant le numéro de la 2<sup>nd</sup> U.A
  - La table FAT permet également de maintenir à jour la liste des U.A non occupées

SYR1-L3 Info &amp; Miage

35

## 4. Organisation d'un SGF FAT

- Rappel : Unité d'Allocation = *cluster*
  - On repère un cluster par son numéro,
  - On peut facilement retrouver l'adresse disque du cluster à partir de son numéro
- Le *boot sector* contient des infos sur
  - Le type de SGF (FAT16, FAT 12, etc.)
  - Le nombre et la taille des clusters
  - La taille de la FAT
- La *File Allocation Table* :
  - Elle contient autant d'entrées qu'il y a de clusters disponibles sur le SGF
  - Sa taille dépend donc de la taille du disque et de la taille des clusters

boot sector  
(réservé)

FAT

Cluster n°2  
Cluster n°3  
Cluster n°4  
Cluster n°5Cluster n°N-2  
Cluster n°N-1  
Cluster n°N

SYR1-L3 Info &amp; Miage

36

## 4. Organisation d'un SGF FAT

- Quelles valeurs pour une entrée de la FAT ?
  - Dépend du type (FAT12, FAT16, FAT32)

| FAT12 | FAT16  | FAT32      | Description                                         |
|-------|--------|------------|-----------------------------------------------------|
| 0x000 | 0x0000 | 0x0000000  | Cluster libre                                       |
| 0x001 | 0x0001 | 0x0000001  | Réservé (utilisé)                                   |
| 0x002 | 0x0002 | 0x0000002  | Utilisé (pointe sur le prochain cluster du fichier) |
| -     | -      | -          |                                                     |
| 0xFE7 | 0xFFEF | 0xFFFFFE7  | Mauvais cluster ou cluster réservé                  |
| 0xFF7 | 0xFFFF | 0xFFFFFFFF | Marqueur de fin de fichier                          |

SYR1-L3 Info &amp; Miage

37

## 4. Exercice

- On se place dans le cadre d'un SGF FAT16
  - Quelles sont les U.A libres de ce SGF ?
  - Quelles sont les U.A utilisées par **toto.txt**
  - Quelles sont les U.A utilisées par **f1.txt**

| Descripteur de fichier |              |            |
|------------------------|--------------|------------|
| Nom externe            | Nombre d'U.A | Entrée FAT |
| toto.txt               | 2            |            |

| Descripteur de fichier |              |            |
|------------------------|--------------|------------|
| Nom externe            | Nombre d'U.A | Entrée FAT |
| f1.txt                 | 11           |            |

|    |      |
|----|------|
| 0  |      |
| 1  |      |
| 2  | 0007 |
| 3  | 0000 |
| 4  | 0000 |
| 5  | FFF8 |
| 6  | 0000 |
| 7  | 000A |
| 8  | 0009 |
| 9  | 000D |
| A  | 0005 |
| B  | 0000 |
| C  | 0000 |
| D  | FFF8 |
| E  | 0000 |
| F  | 0000 |
| 10 | 0000 |
| 11 | 0008 |
| 12 | 0000 |
| 13 | 0000 |

SYR1-L3 Info &amp; Miage

38

## 4. Remarques

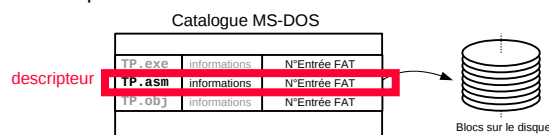
- Remarque 1 :
  - Pour permettre un accès rapide aux données des fichiers, la FAT doit pouvoir être stockée en mémoire vive, ce qui peut poser des problèmes lorsque sa taille est importante !

SYR1-L3 Info &amp; Miage

39

## 4. Catalogue de fichiers sous SGF FAT

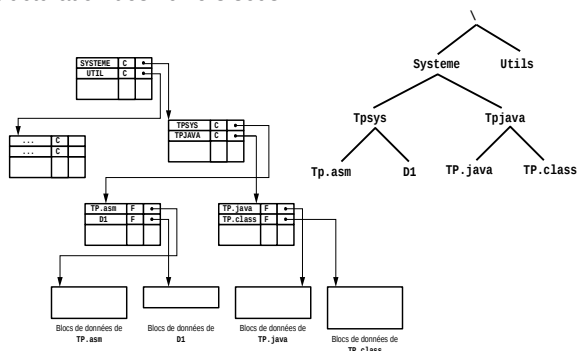
- **catalogue** = table des **descripteurs** de fichiers
  - Quand on veut retrouver les infos d'un fichier à partir de son nom externe, on va parcourir cette table.
  - Chaque **descripteur** contient le **nom simple** du fichier qui lui est associé, plus d'autres informations.
- Exemple :



SYR1-L3 Info &amp; Miage

40

#### Structuration des fichiers sous FAT



SYR1-L3 Info &amp; Miage

41

#### Le SGF FAT n'intègre pas de mécanisme de protection

- Il a été conçu dans les années 80 pour des machines mono utilisateur isolées les unes des autres.
- Son utilisation actuelle est essentiellement pour des dispositifs de stockage amovibles où les problèmes de protections ne sont pas gérés.

SYR1-L3 Info &amp; Miage

42

## I. 5 : Organisation des fichiers sur les SGF Unix

SYR1-L3 Info &amp; Miage

43

### 5. Les *inode* (descripteurs de fichier)

- inode* = descripteur de fichier Unix
  - inode* = Information permanente stockée sur le disque
- Un *inode* contient les informations suivantes
  - L'identifiant du propriétaire du fichier (un numéro d'utilisateur)
  - Le type : fichier de données, répertoire, fichier de périphérique
    - La notion de fichier de périphérique déborde du cadre du cours
  - Les dates de la dernière modification du fichier, du dernier accès au fichier et de la dernière modification de l'*inode*
  - La matrice des droits d'accès (propriétaire, groupe et autres)
  - Le nombre de liens actuellement sur le fichier
  - La taille du fichier en octets
  - La table d'implantation du fichier
- Remarque : l'*inode* ne contient pas de *nom externe* !
  - Permet d'avoir plusieurs noms externes pour un même fichier

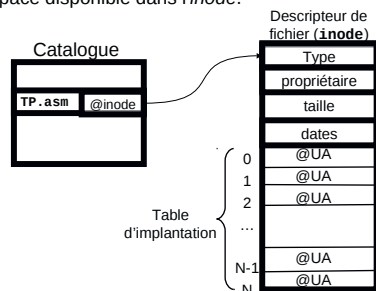
SYR1-L3 Info &amp; Miage

44

### 5. *inode* et table d'implantation

#### Tous les *inode* ont la même taille

- Pour tous les fichiers, la table d'implantation a donc la même taille (on la note N)
- Si le nombre d'UA utilisées par le fichier est  $\ll N$  on sous-utilise l'espace disponible dans l'*inode*.



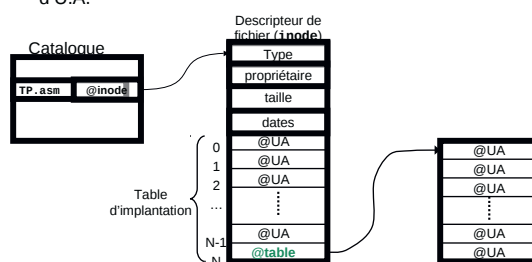
SYR1-L3 Info &amp; Miage

45

### 5. *inode* et table d'implantation

#### Comment gérer le cas où le fichier est trop grand ?

- C.a.d que le nombre d'UA utilisées par le fichier est  $> N$
- On utilise une des cases de la table d'implantation pour stocker l'adresse d'un bloc disque contenant la suite des adresses d'U.A.



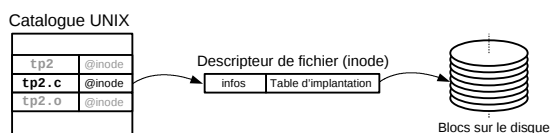
SYR1-L3 Info &amp; Miage

46

### 5. Les catalogues sous Unix

#### catalogue Unix = table de couples (*nom*, *@inode*)

- @inode : c'est un numéro qui permet d'identifier le descripteur de fichier.
- Un même *inode* peut-être associé à plusieurs nom externes (lien non symboliques gérés par la commande **ln** sous Unix)
- Comme c'est le cas pour les *inode*, le catalogue doit être stocké quelque part sur le disque.



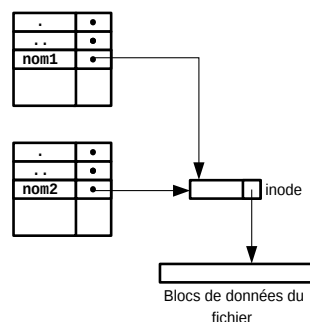
- On a de fait une séparation *nom externe/descripteur*

SYR1-L3 Info &amp; Miage

47

### 5. Séparation nom externe/descripteur

- Cette séparation permet de désigner un même fichier (au sens espace disque) par plusieurs noms différents

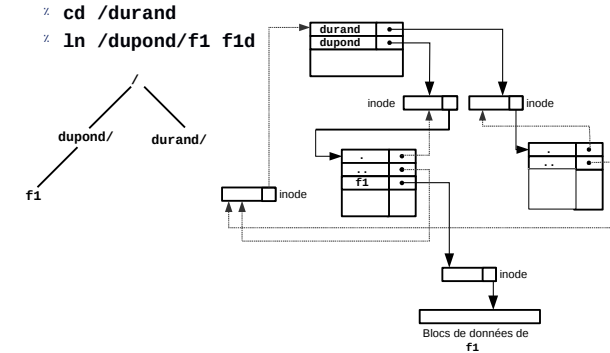


SYR1-L3 Info &amp; Miage

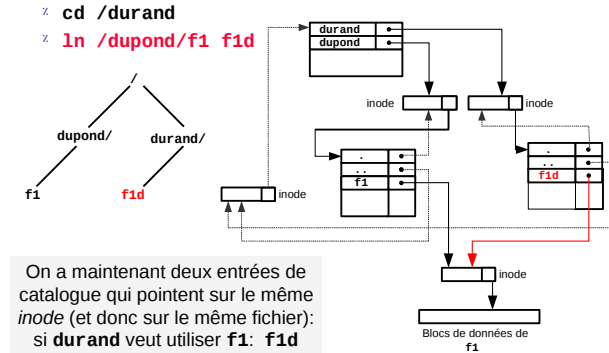
48



Création d'un lien (commande UNIX ln)



Création d'un lien (commande UNIX ln)



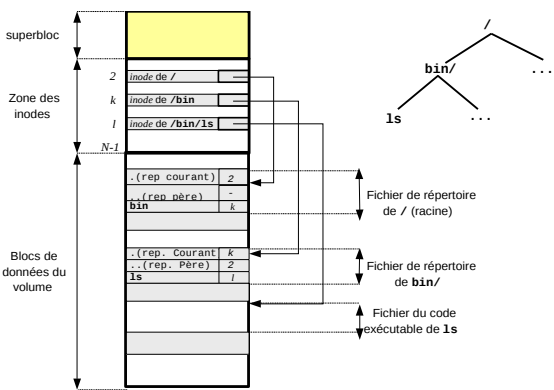
On a maintenant deux entrées de catalogue qui pointent sur le même *inode* (et donc sur le même fichier): si *durand* veut utiliser *f1*: *f1d*

5. Organisation d'un volume UNIX

Quelque définitions

- volume : disque (ou partie de disque) organisée comme une suite d'U.A. de taille fixe sur lesquelles sont enregistré un ensemble de fichiers constituant une hiérarchie.
  - superbloc : zone réservée sur le disque contenant une description de volume (listes des U.A. libres, etc.)
  - inode : descripteur de fichier (contient entre autres la table d'implantation du fichier).
  - inode-list : liste des *inode* libres (stockée dans la zone de disque allouée au *superbloc*).
- Remarques :
- Les répertoires sont vus comme des fichiers (ils ont leur propre *inode* comme les fichiers de données).

5. Organisation d'un volume UNIX



5. Le superbloc

- Structure contenant les informations suivantes :
- taille du volume contenant le *superbloc*
  - nombre d'U.A. libres dans le volume
  - liste des UA libres
  - index de la prochaine UA libre dans la liste
  - taille de la listes des *inode*
  - nombre d'*inode* libres
  - listes des *inode* libres
  - index du prochaine *inode* sur la listes des *inode* libres
  - ...
- Il permet d'avoir une vue globale du volume

5. Droits d'accès sous Unix

- Pour chaque fichier/répertoire, on donne des droits pour
- Le propriétaire (*user*)
  - Le groupe du propriétaire (*group*)
  - Les autres (*others*)
- Plusieurs types de droits
- Pour les fichiers : *r=lire*, *w=écrire*, *x=exécuter*
  - Pour les répertoires : *r=lister le contenu*, *w=écrire*, *x=accès (cd)*
- On utilise 9 bits pour représenter ces droits d'accès
- Cette information fait partie du descripteur de fichier (*inode*)

5. Droit d'accès sous Unix

```
%:/share/l3info/SYS2/TP-SGF> ls -ail
total 176
permissions
4600957 drwxrwsr-x 7 sderrien ens 4096 fev 7 17:41 ./
4600953 drwxrwsr-x 7 sderrien ens 4096 fev 16 08:35 ../
4600970 drwxrwsr-x 4 sderrien ens 4096 fev 7 17:41 doc/
2966648 drwxrwsr-x 2 sderrien ens 4096 fev 7 17:41 es/
2966641 drwxrwsr-x 2 sderrien ens 4096 fev 7 17:41 esbase/
4600968 -rwxrwxr-x 1 sderrien ens 645 fev 9 2005 makefile*
2966637 drwxrwsr-x 2 sderrien ens 4096 fev 7 17:41 sgf/
4600958 -rwxrwxr-x 1 sderrien ens 384 jan 28 2005 ti1*
2966628 drwxrwsr-x 2 sderrien ens 4096 fev 7 17:41 tests/
4600959 -rwxrwxr-x 1 sderrien ens 148 mars 2 2004 ti1*
4600960 -rwxrwxr-x 1 sderrien ens 256 mars 2 2004 ti2*
4600961 -rwxrwxr-x 1 sderrien ens 404 jan 28 2005 ti3*
4600962 -rwxrwxr-x 1 sderrien ens 256 mars 2 2004 ti4*
4600963 -rwxrwxr-x 1 sderrien ens 768 mars 2 2004 ti5*
4600964 -rwxrwxr-x 1 sderrien ens 129 mars 2 2004 ti6*
4600965 -rwxrwxr-x 1 sderrien ens 256 mars 2 2004 ti7*
4600966 -rwxrwxr-x 1 sderrien ens 258 mars 2 2004 ti8*
4600967 -rwxrwxr-x 1 sderrien ens 16384 mars 2 2004 ti9*
4600969 -rwxrwxr-x 1 sderrien ens 303 fev 1 2005 zip.sh*
n°inode
```

I . 6 : Comparaison Unix/FAT

- L'accès au contenu d'un fichier se fait en deux étapes
1. Recherche du fichier dans le catalogue à partir de son nom externe.

2. Accès au contenu du fichier grâce à la table d'implantation stockée dans le descripteur.
- Les catalogues UNIX occupent moins d'espace disque.
- Le catalogue est plus petit, on a donc moins d'informations à lire sur le disque et l'étape (1) est plus rapide que sur FAT.

Mais il faut faire une E/S physique pour récupérer le contenu du descripteur (ne n'est pas le cas pour FAT)

Car le descripteur de fichier est déjà dans le catalogue

En moyenne, on est quand même gagnant en nombre d'E/S physiques par rapport à FAT.

- Unix permet le partage d'un descripteur
- Un même descripteur peut être utilisé par plusieurs entrées du catalogue ayant des noms différents (synonymes)
- f1

f1bis

Données du fichier sur disque
- Deux noms
- un seul descripteur  
(une seule table d'implantation)

Partie II : réalisation des transferts

- II.1 Désignation d'un fichier
- II.2 Fonctions logiques de transfert
- II.3 Mise en œuvre des accès : principes
- II.4 Mise en œuvre des accès : implémentation

Introduction

istic

- Dans la première partie du cours nous avons
- Étudié la manière dont les fichiers sont stockés sur disque

Comparé les approches Unix et MS-FAT
- Mais comment se font les accès à ces fichiers ?
- L'objectif est de présenter des principes de base

MAIS, qui permettent de réaliser des E/S logiques efficaces.

En utilisant des tampons d'E/S

En utilisant un cache de bloc d'E/S

Plan

istic

1. Liaison nom logique/nom externe

1. Nommage des fichiers

2. Liaison à l'exécution/par commande
2. Les fichiers flots : organisation et accès

1. Types de fichiers

2. Accès aux fichiers flots

3. Accès formatés
3. Mise en œuvre des accès : principes

1. Notion d'E/S logique

2. Le tampon d'entrée/sortie

3. Le cache de blocs d'E/S
4. Mise en œuvre des accès : implémentation

1. Le bloc de contrôle de fichiers

2. Exemple de fonctionnement

3. Mise en œuvre des BCF sous Unix

II . 1 : Liaison nom logique / nom externe

1. Définition : Nom logique

istic

- Nom externe : identifiant du fichier sur le disque
- Fourni par l'utilisateur lors de la création du fichier

Nom permanent et global utilisé par le SGF

Stocké sur disque dans un descripteur de fichier (FAT) et/ou dans un catalogue (Unix)
- Nom logique : identifiant du fichier dans un programme
- Les prog. ne manipulent pas directement le nom externe

Ils utilisent un nom logique, défini seulement dans le prog.

Le nom logique est temporaire et est lié à un environnement d'exécution particulier.

|             | Utilisation       | Durée de vie           |
|-------------|-------------------|------------------------|
| nom externe | Par le système    | Illimitée*             |
| nom logique | Dans le programme | Exécution du programme |

1. Liaison par procédure système

istic

- Principe de fonctionnement
- Le programme détermine le nom externe du fichier à utiliser (par ex. en demandant à l'utilisateur de saisir son nom au clavier)

Le programme fait un appel système en passant en paramètre le nom externe et obtient un nom logique en retour de l'appel.
- Exemple (en langage C)
- ```
char nom_externe[20];
FILE *nom_logique;

void main() {
    scanf("%s",nom_externe);
    nom_logique = fopen(nom_externe, "r" );
    ...
    fclose(nom_logique);
}
```


1. Liaison par commande

- Principe de fonctionnement
 - La liaison est faite avant ou juste à l'appel du programme
 - Elle concerne en général l'E.S ou la S.S
- Exemple (langage de commande Unix) :
 - Deux fichiers standard **stdin** et **stdout** (noms logiques)
 - Le fichier **stdin** désigne l'E.S, **stdout** désigne la S.S
 - Les redirections (>, <, >>) permettent d'associer des noms externes aux fichiers standard.
 - La commande **tp2 < test.txt > res.txt** va effectuer la liaison entre :
 - Le nom logique **stdin** et le nom externe **test.txt**,
 - Le nom logique **stdout** et le nom externe **res.txt**,

II . 2 : Les fichiers flot : organisation et accès

2. Différents types d'Entrées/Sorties

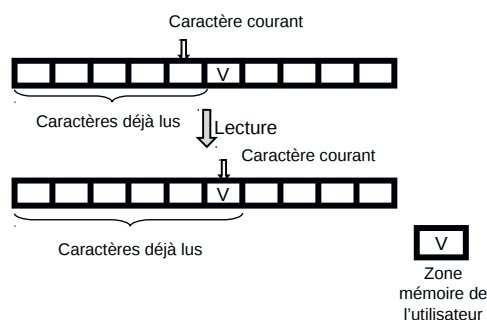
- Entrées/sorties de caractères
 - Par *E/S caractère* on entend en fait E/S sur des octets (on lit/écrit octet par octet dans le fichier).
 - Attention* : faire des E/S caractère ne veut donc pas dire que l'on manipule des fichiers texte (on peut faire des E/S caractère sur un fichier binaire).
- Entrées/sorties formatées
 - Exemple : lire/écrire des entiers/réels directement sous forme textuelle (c.-a.-d. comme un suite de codes ASCII).
 - On effectue simplement une conversion (ex : entiers ⇔ chaîne)
- Entrées/sorties d'articles
 - article* = type structuré (taille fixe), avec parfois un champ *clé*
clé = identifiant unique sur lequel il existe une relation d'ordre
 - Exemple : article *étudiant* = (*n°étudiant*, *nom*, *age*, etc.)
 - Très utilisé dans les Systèmes de Gestion de Bases de Données

2. Fichiers *flot* (ou *fichiers caractères*)

- Un fichier flot peut être vu comme une *file d'octets*
 - On accède à ces octets de manière séquentielle
 - Ces fichiers sont utilisables en *lecture ou en écriture*
 - On ne peut pas lire et écrire dans un fichier en même temps
 - Plusieurs programmes peuvent lire un fichier en même temps
- Quelles sont les opérations possibles sur ces fichiers ?
 - Se positionner dans la file (début, fin)
 - Lire le caractère suivant dans la file
 - Écrire un caractère en fin de file
- Nous n'étudierons que les fichiers flots
 - C'est le plus simple et le plus utilisé des types de fichiers
 - Disponible pour tous les systèmes (souvent le seul disponible !)
 - Issu des premières machines informatiques avec stockage sur bandes magnétiques (donc avec un accès séquentiel)

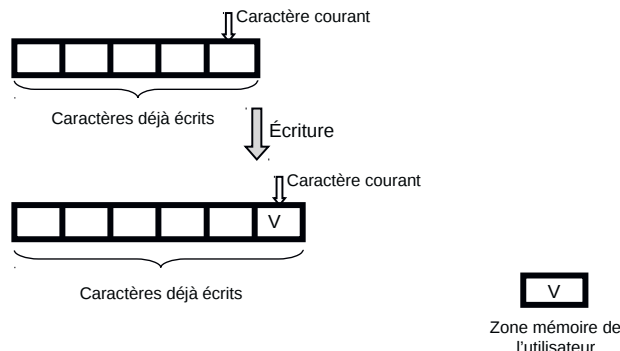
2. Fichier de type flot

- Exemple de lecture



2. Fichier de type flot

- Exemple d'écriture



2. Fichiers *directs* et *séquentiels indexés*

- Il existe d'autres types de fichiers
 - Fichiers direct* : le fichier est vu comme un tableau d'*articles*. On accède à un article à partir d'un index.
 - Fichier séquentiel indexé* : le fichier est vu comme un ensemble de couples (*clé*, *article*) et permet un accès rapide à un article à partir de sa clé.
- Principales utilisations
 - Les *Fichiers séquentiels indexés* sont très utilisés pour stocker les bases de données car ils permettent une recherche très rapide ($O(\log n)$) de l'information à partir d'une clé.
 - Ces types de fichiers sont gérés par des Systèmes de Gestion de Base de Données (SGBD) qui sont souvent construits « au dessus » de la couche SGF du système d'exploitation.

2. Accès aux fichiers de type flot (C/Unix)

- Fonctions définies dans le fichier **<stdio.h>**
 - #include <stdio.h>**
- Principales fonctions d'accès (rappel)
 - FILE* fopen(char* nomext, char* mode)** : initialise les accès sur le fichier de nom externe **nomext** en utilisant le mode spécifié dans la chaîne **mode** ("r" = lecture, "w" = création)
 - int fgetc(FILE* f)** : lit un octet dans le fichier logique **f** et retourne la valeur lue (convertie en **int**, retourne la constante **EOF** en cas de fin de fichier).
 - int fputc(char c, FILE* f)** : ajoute un octet **c** dans le fichier logique **f** (en fin de fichier). La fonction retourne **EOF** en cas d'erreur.
 - void fclose(FILE* f)** : ferme les accès sur le fichier logique passé en paramètre.

2. Accès aux fichiers de type flot (C/Unix) istic

Accès formatés aux fichiers

- `int fscanf(FILE* f, char* format, a1, a2, ..., an)`
Même fonctionnement que pour `scanf()`, mais les données sont lues à partir du fichier logique `f` au lieu de l'E.S.
- `int fprintf(FILE* f, char* format, e1, e2, ..., en)`
Même fonctionnement que pour `printf()`, mais on écrit les données dans le fichier `f` plutôt que sur la S.S.

Remarques :

- Ces fonctions d'E/S accèdent au fichier octet par octet. Le fichier ne contient que des octets dont les valeurs correspondent à des codes ASCII.
- Elles effectuent simplement un traitement particulier sur ces octets/code ASCII en fonction du type de format utilisé (ex: syntaxe des réels).

SYR1-L3 Info & Miage

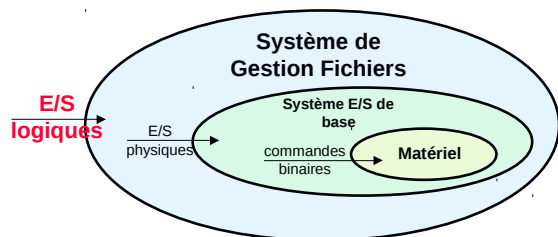
73

II . 3 : Mise en œuvre des accès : principes

SYR1-L3 Info & Miage

74

3. MEO des accès : couches d'E/S système istic



Dans ce chapitre on va étudier les E/S logiques...

SYR1-L3 Info & Miage

75

3. MEO des accès : E/S physique istic

- ▮ Une E/S physique accède au périphérique de stockage
 - C'est un accès coûteux en temps d'exécution (~10 ms), on cherche à en faire le plus petit nombre possible.
- ▮ Les fonctions E/S physique opèrent sur des blocs d'E/S
 - À chaque fois, on lit/écrit un bloc d'E/S entier, la taille d'un bloc d'E/S étant définie par le S.E
 - Ce sont des fonctions fournies par le système d'E/S physique
 - `int readblock(int addr_sector, char* buffer)`
 - `int writeblock(int addr_sector, char* buffer)`

SYR1-L3 Info & Miage

76

3. MEO des accès : E/S logique istic

- ▮ E/S logique = les E/S utilisateur dans un programme
 - Exemple : `char c = fgetc(fichier);`
- ▮ Concernent une quantité d'info définie par l'utilisateur
 - Un ou plusieurs caractères.
- ▮ En général, la taille des infos échangées lors d'une E/S logique est très inférieure à la taille d'un bloc d'E/S
 - caractère = 1 octet, bloc d'E/S > 512 octets

On souhaite minimiser le rapport entre le nombre d'E/S physiques et le nombre d'E/S logiques

SYR1-L3 Info & Miage

77

3. MEO des accès : le tampon d'E/S istic

- ▮ Problématique
 - Les *fonctions d'E/S logiques* offertes à l'utilisateur concernent des caractères (fichier flot).
 - Les *fonctions d'E/S physiques* se font au niveau bloc d'E/S, et pas au niveau article/caractère.
 - En règle générale la taille du bloc d'E/S ≠ taille des articles)
- ▮ Comment faire le lien entre E/S physiques et logiques ?
 - On va voir comment mettre en œuvre ces accès, en utilisant une mise en œuvre naïve, puis en l'améliorant grâce à l'utilisation d'une mémoire tampon d'E/S et d'une mémoire cache.

SYR1-L3 Info & Miage

78

3. MEO des accès : approche naïve istic

- ▮ **Principe** : à chaque fois qu'on fait une E/S logique, on fait une (ou plusieurs E/S physiques).
 - **Exemple 1** : pour la lecture logique d'un caractère, on va
 1. Lire sur le disque le bloc contenant le caractère (1 E/S physique)
 2. Copier le caractère vers la zone mémoire utilisateur
 - **Exemple 2** : pour l'écriture logique d'un caractère, on va
 1. Lire le bloc contenant ce caractère (1 E/S physique)
 2. Modifier la copie du bloc en mémoire
 3. Réécrire le bloc sur le disque (1 E/S physique)
- ▮ **Inconvénients**
 - E/S physique très coûteuse en temps de calcul (~ ms)
 - Une même information est souvent lue plusieurs fois
 - C'est une approche très peu efficace

SYR1-L3 Info & Miage

79

3. MEO des accès : le tampon d'E/S istic

- ▮ **Principe** :
 - Pour réduire le nombre d'E/S physiques, on va utiliser une mémoire tampon, on ne fera une E/S physique que lorsque le tampon sera vide (ou plein)
- ▮ **Exemple** : cas de la lecture (de caractères)
 - Les E/S logiques vont lire dans une **mémoire tampon**
 - Cette mémoire tampon contient le **dernier bloc lu** sur le disque
 - La lecture du caractère se fait dans le tampon
 - Si le tampon a été entièrement lu, on lance une nouvelle E/S physique pour remplir le tampon avec un nouveau bloc d'E/S
- ▮ **Remarque** :
 - Le fonctionnement pour l'écriture est symétrique : on écrit dans le tampon. Quand celui-ci est plein, on l'écrit sur disque

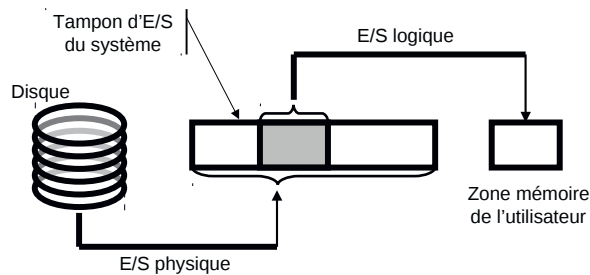
SYR1-L3 Info & Miage

80

3. MEO des accès : le tampon d'E/S istic

Principe

- On stocke le bloc en cours de lecture par le programme dans une **mémoire tampon** système.



SYR1-L3 Info & Miage

81

3. MEO des accès : le tampon d'E/S istic

Mise en œuvre des tampons d'E/S

- Pour un programme, on a autant de tampons d'E/S que de fichiers logiques ouverts.
- Le tampon est **alloué dynamiquement dans la mémoire**, en général lors de l'appel système effectuant la liaison.
- La gestion de ces tampons est faite par le système d'exploitation au travers des fonctions d'E/S logiques

Limites de l'approche

- Parfois, la réutilisation apparaît au niveau blocs
 - Exemple : un même fichier relu plusieurs fois dans un même programme.
- N'exploite pas assez la localité spatiale
 - Par localité « spatiale » on entend le fait qu'on lit souvent des blocs d'E/S contigus sur le disque

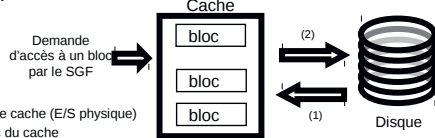
On utilise souvent un cache en plus des tampons d'E/S

SYR1-L3 Info & Miage

82

3. Le cache d'E/S istic

- On accède souvent à des blocs contigus sur le disque,
 - Un fichier/répertoire a souvent ses blocs de données placés proche les uns des autres sur le disque.
 - Il peut être intéressant de faire des E/S physiques de plusieurs blocs.
- Plusieurs processus peuvent accéder à un fichier en même temps
 - On va relire plusieurs fois les mêmes blocs
- Dans tous ces cas, le tampon d'E/S a une efficacité limitée
 - La réutilisation se fait au niveau bloc pas à l'intérieur d'un bloc
 - On va donc essayer de garder dans une **mémoire cache** les blocs souvent utilisés.



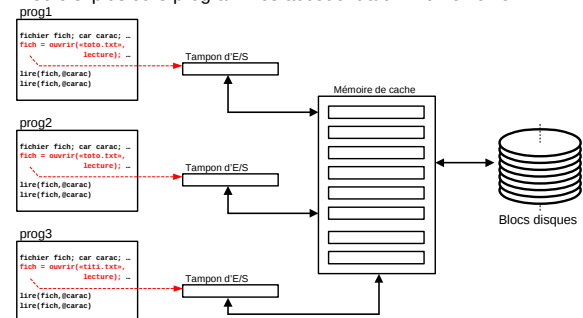
SYR1-L3 Info & Miage

83

3. Le cache istic

La cache est stocké dans la mémoire système

- Il peut être partagé par tous les programmes/processus
- Utile si plusieurs programmes accèdent à un même fichier



SYR1-L3 Info & Miage

84

3. Fonctionnement du cache istic

Organisation du cache en mémoire

- On réserve une partie de la **mémoire système** pour stocker les blocs de cette mémoire cache.
- Cette mémoire cache est composée de cases, une case de cache contient m blocs d'E/S contigus (« souvent » $m=1$)
- On fait toujours des E/S d'une case entière (évite d'attendre le positionnement de la tête de lecture du disque à chaque fois)

Que se passe-t-il pour une E/S sur un bloc en cache ?

- Pas besoin d'E/S physique, on lit/écrit dans la case du cache
 - On fait simplement une copie de mémoire à mémoire
- Pour une écriture, le contenu du disque n'est plus « à jour »
 - Il faudra remettre le contenu du disque à jour lorsque l'on remplacera la case du cache.

SYR1-L3 Info & Miage

85

3. Fonctionnement du cache istic

Que faire quand on doit lire un bloc absent du cache ?

- Il faut trouver une case vide, ou en libérer une
- Le plus souvent, il faut faire une **libération de case**
- Le choix de la case à libérer est fait selon une algorithmique qui implémente une **politique de gestion du cache** (cf. plus loin)
- On **libère une case** en écrivant son contenu sur le disque
 - ATTENTION** : on ne fait une écriture que si des données de la case ont été modifiées (on utilise pour cela un marqueur).
 - On met alors à jour le contenu du disque à partir de ces données, ce qui implique de faire m E/S physiques (m = taille d'une case).
- Il faut ensuite lire sur le disque le(s) bloc(s) manquant(s)
 - m E/S physiques et copie du/des bloc(s) dans la case du cache
- Coût : m lectures physiques + éventuellement m écritures physiques

SYR1-L3 Info & Miage

86

3. Fonctionnement du cache istic

Que faire quand on doit écrire un bloc absent du cache ?

- Il faut trouver une case vide, ou en libérer une
 - On suit le même principe que pour la lecture dans le cache
- On lit (sur le disque) le contenu de la case dans laquelle on doit écrire le bloc.
 - Le cache ne sait faire que des E/S physiques de m blocs
- Le contenu du disque n'est alors plus « à jour »
- Le coût en termes d'E/S est le même que dans le cas de la lecture d'un bloc absent

Lors de la fermeture d'un fichier

- Tous les blocs modifiés du cache sont écrits sur le disque
 - Permet de s'assurer que le contenu du disque est bien à jour
- En pratique, on le fait à intervalle régulier
 - Limite les problèmes en cas de panne

SYR1-L3 Info & Miage

87

3. Mise en œuvre du cache istic

Que faut-il pour mettre en œuvre un cache ?

- Une structure de données décrivant le contenu du cache
 - Un tableau de cases (contenant les blocs d'E/S)
 - Un tableau indiquant les n° des blocs contenus dans ces cases.
- Une politique du cache qui doit satisfaire deux critères :
 - Éviter de libérer une case qui a de fortes chances de resservir
 - Car sinon il faudra la recharger => E/S physique
 - Éviter de libérer une case contenant un bloc modifié
 - Sinon il faut la sauver alors qu'elle pourrait encore être modifiée

Exemples de politique de gestion de cache

- LRU (*Least Recently Used*) : on libère la case qui a servi le moins **récemment**
- LFU (*Least Frequently Used*) : on libère la case qui a servi le moins **fréquemment**

SYR1-L3 Info & Miage

88

3. Exemple de fonctionnement du cache

- Caractéristiques du cache
 - Chaque case contient 4 blocs d'E/S, le cache contient 8 cases.
 - La politique de gestion du cache est « Least Recently Used »
 - On utilise un marqueur pour repérer le bloc le moins récemment utilisé.
- Caractéristiques du disque
 - L'accès à des blocs d'E/S contigus sur le disque permet d'améliorer le temps moyen de lecture d'un bloc car on n'a pas à déplacer la tête entre deux accès à des blocs contigus.
 - On va utiliser donc la formule $T_{E/S} = 8 + 4 * (nb_{blocsE/S})$
(8 = temps moyen posit., 4 = temps lect. bloc)
 - Exemples :
 - Temps de lecture moyen d'un bloc d'E/S contigus = 12 ms
 - Temps de lecture moyen de 2 blocs d'E/S contigus = 16 ms
 - Temps de lecture moyen de 4 blocs d'E/S contigus = 24 ms

SYR1-L3 Info & Miage

89

3. Exemple de fonctionnement du cache

- Caractéristiques du SGF
 - Le fichier exécutable prog1 occupe les blocs 7,8,9
 - Le fichier exécutable prog2 occupe les blocs 20,21,22
 - Le fichier f1.txt occupe les blocs 12,45,46,23,14,89,72,17
 - Le fichier f2.txt occupe les blocs 44,13,15,16, 31,61
 - Soit les programmes prog1 et prog2 (et leur code C ci-dessous)
- Donner l'état du cache après exécution de prog1, prog2

```
/****** prog1.c *****/
FILE* f;
char c;

f = fopen("f1.txt", "r");
int res = fgetc(f);
while(res!=EOF) {
    c=(char) res;
    res = fgetc(f);
}
```

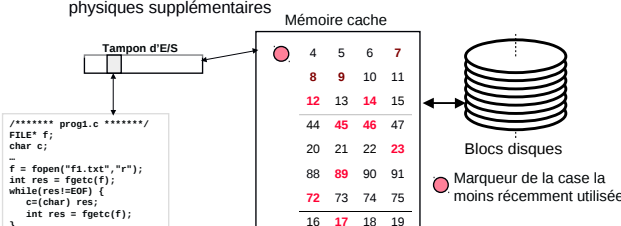
```
/****** prog2.c *****/
FILE* f;
char c;

f = fopen("f2.txt", "r");
int res = fgetc(f);
while(res!=EOF) {
    c=(char) res;
    res = fgetc(f);
}
```

SYR1-L3 Info & Miage

90

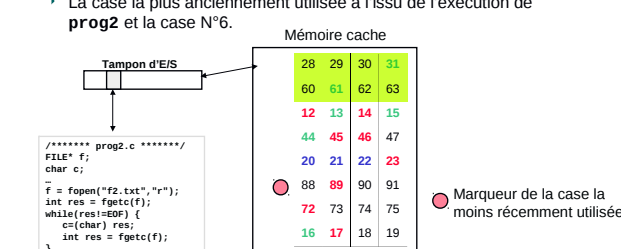
3. Exemple de fonctionnement du cache

- À l'issue de l'exécution de prog1 :
 - 8 lignes du cache chargées, avec une durée de = $8*24=192$ ms
 - Sans le cache il aurait seulement fallu $(3+8)*12=132$ ms
 - La case du cache la plus anciennement utilisée et la case n°1
 - Si on re-exécute prog1 immédiatement, il n'y aura pas d'E/S physiques supplémentaires
- 

SYR1-L3 Info & Miage

91

3. Exemple de fonctionnement du cache

- À l'issue de l'exécution de prog2 :
 - Le contenu du fichier exécutable **prog2** est déjà dans la cache, donc son chargement ne nécessite pas d'E/S physique
 - Pour **f2.txt** on a du recharger les cases N°1 et N°2 du cache, pour un coût en temps de $2*24=48$ ms (sans cache il faut 108 ms)
 - La case la plus anciennement utilisée à l'issue de l'exécution de **prog2** et la case N°6.
- 

SYR1-L3 Info & Miage

92

II . 4 : Mise en œuvre des accès : implémentation

SYR1-L3 Info & Miage

93

4. Le Bloc Contrôle Fichier (BCF)

- Quand un programme fait une E/S sur un fichier, il utilise :
 - Des *informations permanentes* (issues du descripteur) pour pouvoir par exemple accéder à la *table d'implantation*.
 - Des informations particulières *liées à une exécution*, par exemple pour savoir où on est positionné dans le fichier.
- Ces infos sont regroupées dans le Bloc Contrôle Fichier
 - Le BCF contient donc des infos concernant un fichier ouvert
 - C'est ce BCF qui est désigné par les opérations d'E/S

Informations d'accès	Informations permanentes
◆ Adresse du tampon E/S	◆ Propriétaire
◆ Numéro caractère courant	◆ Droits d'accès
◆ Numéro bloc d'E/S courant	◆ Table d'implantation
◆ Position dans le tampon	◆ Etc.

SYR1-L3 Info & Miage

94

4. BCF et fichiers flots

- Fichier = file de caractères
 - Ordre sur l'accès aux éléments du fichier
 - Donc ordre a priori sur l'accès aux blocs
 - L'utilisation d'un tampon d'E/S permet de réduire très nettement le nombre d'E/S physiques dans ce cas.
 - Opération de lecture
 - Pas besoin de relire physiquement un bloc tant que l'on ne l'a pas épuisé
 - Opération d'écriture
 - Pas besoin d'écrire physiquement un bloc tant que l'on n'a pas rempli le bloc courant
- Informations d'accès à mettre dans le BCF
 - Position dans le fichier
 - Numéro du dernier bloc lu ou écrit
 - Numéro du prochain caractère à lire dans le tampon

SYR1-L3 Info & Miage

95

4. Le Bloc contrôle fichier : utilisation

- À l'ouverture du fichier
 - Allocation d'un BCF dans la table des BCF du système.
 - Initialisation des informations d'accès temporaire du BCF (position dans le fichier, allocation du tampon d'E/S)
 - Initialisation des informations d'accès permanentes (recherche sur le disque du descripteur de fichier pour obtenir la taille du fichier, sa table d'implantation, etc.)
- Lors d'une lecture logique :
 - Modification de la position dans le fichier, de la position dans le tampon et éventuellement du n° de bloc courant
 - Si on doit passer au bloc suivant :
 - On lit la table d'implantation pour retrouver le bloc de données qui contient la suite des données du fichier
 - On copie le contenu de ce bloc dans le tampon d'E/S

SYR1-L3 Info & Miage

96

■ Lors d'une écriture logique :

1. Modification de la position dans le fichier, de la position dans le tampon et éventuellement du n° de bloc courant
2. Si le tampon d'E/S est plein :
 - a) On détermine le bloc disque supposé contenir les données écrites dans le tampon.
 - b) On copie le contenu du tampon d'E/S vers ce bloc disque

■ À la fermeture du fichier

- Mise à jour du contenu du disque (en cas d'écriture)
 - Sauvegarde éventuelle du contenu du tampon d'E/S sur disque
 - Mise à jour du descripteur de fichier sur le disque
- Libération de la mémoire occupée par le tampon d'E/S
- Libération du BCF

■ Il ne contient pas forcément toutes les informations

- Il peut n'utiliser qu'un pointeur sur l'entrée du catalogue ou du descripteur (là où sont les informations)

■ En résumé

- Le BCF sert d'interface entre le **programme utilisateur** et le **système de gestion de fichiers**.
- Sa mise en œuvre peut varier d'un système à l'autre, mais le principe reste le même.
- Le TP SGF va consister à mettre en œuvre la gestion du BCF pour des fichiers de caractères.

4. BCF : illustration du fonctionnement

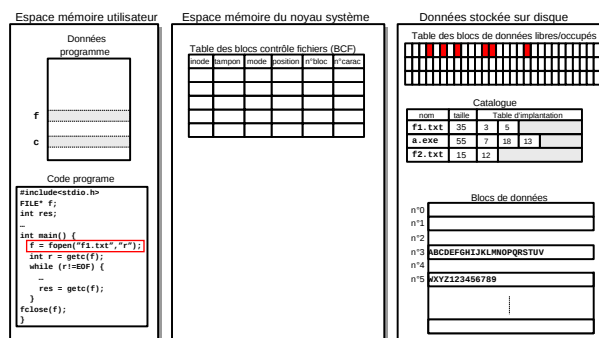
■ On suppose que :

- Le programme effectue des E/S logiques caractères sur un fichier ouvert en mode *lecture*.
- Les secteurs et les blocs d'E/S occupent 22 octets. Il n'y a pas de mémoire cache pour les blocs d'E/S
- Le catalogue contient des descripteurs de fichier (idem FAT)
- Le code programme exécuté est donné ci-dessous

```
#include <stdio.h>
FILE* f;
int res;
...
int main() {
    f = fopen("f1.txt", "r");
    res = getc(f);
    while (r!=EOF) {
        ...
        res = getc(f);
    }
    fclose(f);
}
```

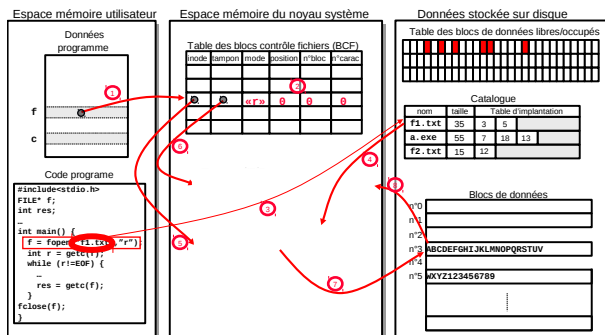
4. Bloc contrôle de fichier : fonctionnement

1) Avant l'initialisation des accès par fopen(...)



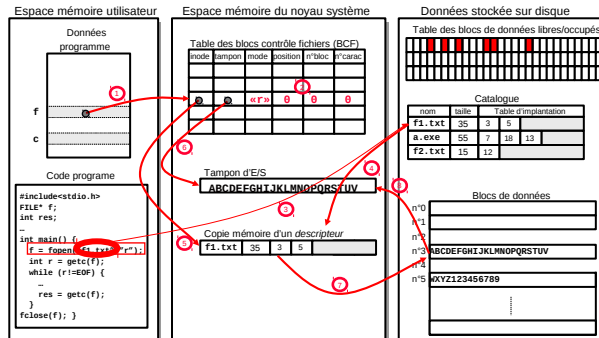
4. Bloc contrôle de fichier : fonctionnement

2) Pendant l'initialisation des accès par fopen(...)



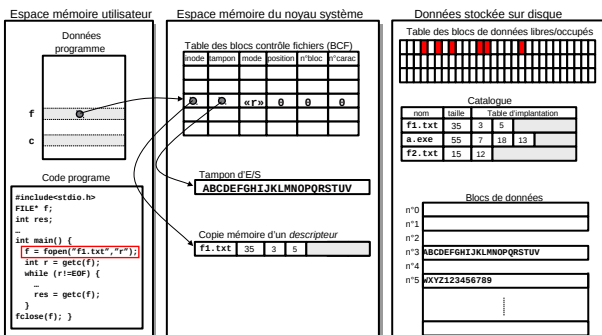
4. Bloc contrôle de fichier : fonctionnement

2) Résumé des accès par fopen(...)

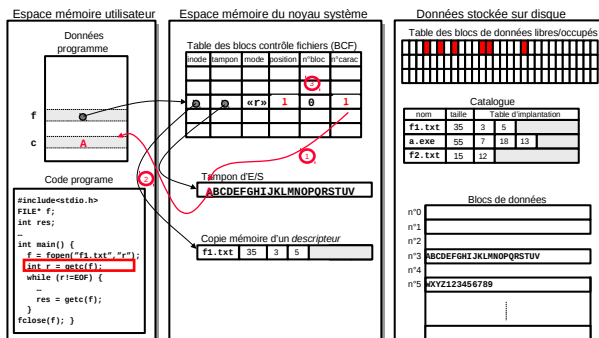


4. Bloc contrôle de fichier : fonctionnement

3) À l'issue de l'initialisation des accès par fopen(...)

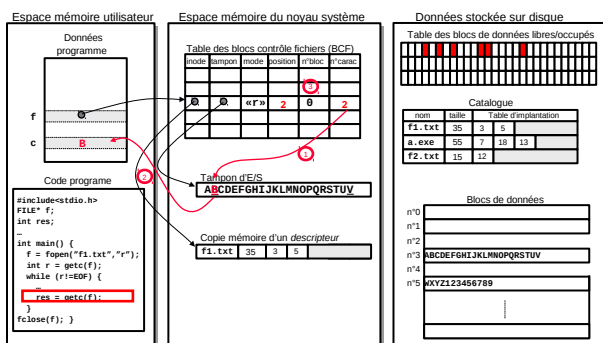


4. Bloc contrôle de fichier : fonctionnement

4) Lecture du 1^{er} caractère ('A') par l'appel à getc(f)

4. Bloc contrôle de fichier : fonctionnement

4) Lecture du 2nd caractère ('B') par l'appel à getc(f)

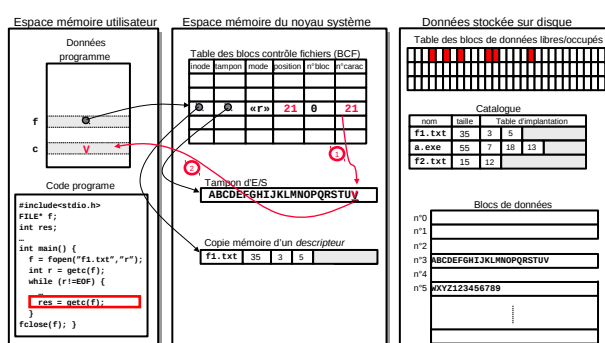


SYR1-L3 Info & Miage

105

4. Bloc contrôle de fichier : fonctionnement

4) Lecture du 22^{ème} caractère ('V') par l'appel à getc(f) (début)

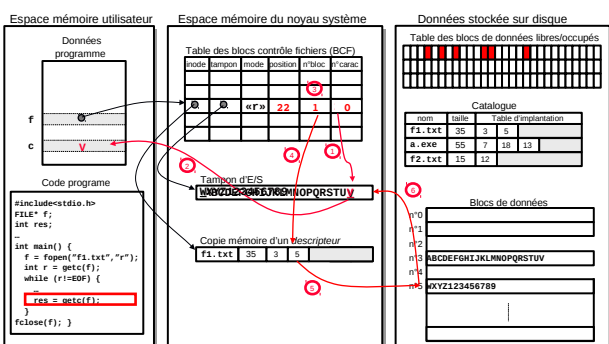


SYR1-L3 Info & Miage

106

4. Bloc contrôle de fichier : fonctionnement

4) Lecture du 22^{ème} caractère ('V') par l'appel à getc(f) (fin)

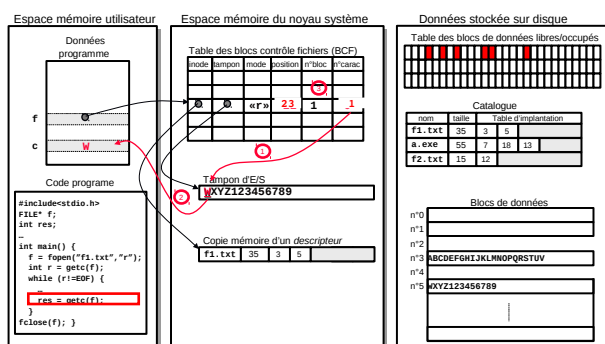


SYR1-L3 Info & Miage

107

4. Bloc contrôle de fichier : fonctionnement

4) Lecture du 23^{ème} caractère ('W') par l'appel à getc(f)



SYR1-L3 Info & Miage

108

4. Exemple de MEO des BCF : Unix

- UNIX est un S.E multi-tâches/Multi-Utilisateur
 - ▶ Plusieurs programmes utilisateur sont exécutés en même temps (ils se partagent le temps de calcul de la machine).
 - On appelle ces programmes des processus
 - ▶ Cet aspect des S.E. (et sa mise en œuvre) sera abordé en M1
 - Module Système de Gestion des Processus
- Conséquence sur la gestion des accès aux fichiers
 - ▶ Il peut y avoir plusieurs programmes indépendants qui accèdent à un même fichier sur le disque.
 - ▶ Il faut s'assurer qu'il n'y a pas d'incohérences (exemple : fichier du disque ouvert en lecture et en écriture en même temps)
 - ▶ On souhaite éviter de répliquer inutilement des informations (exemple : deux copies mémoire d'un même inode)

SYR1-L3 Info & Miage

109

4. Exemple de MEO des BCF : Unix

- On répartit les informations entre le système et le prog.
 - ▶ Les informations pouvant être partagées entre plusieurs utilisateurs sont stockées dans la mémoire système
- Contenu de la *mémoire système* :
 - ▶ On y trouve une copie mémoire des *inode* des fichiers utilisés, c'est la *i-node Table*, dont chaque entrée est une copie d'un *inode* stocké sur le disque.
 - ▶ On y trouve la table des fichiers ouverts dans le noyau, c'est la *File Table*, chacune de ses entrées est associée à un fichier ouvert et contient les informations d'accès (équivalent au BCF).
- Contenu de la *mémoire utilisateur* :
 - ▶ Chaque processus dispose d'une table qui peut contenir des pointeurs sur des entrées de la *File Table*, c'est la *user file descriptor table*.

SYR1-L3 Info & Miage

110

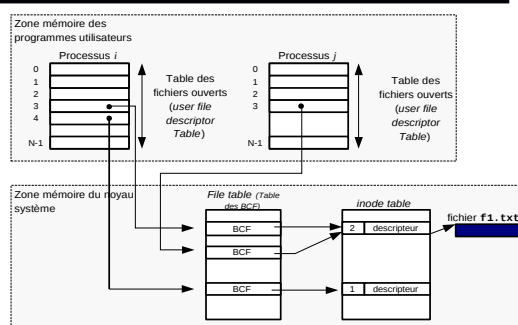
4. Exemple de MEO des BCF : Unix

- Descripteurs de fichier logique sous Unix
 - ▶ Les descripteurs sont en fait des numéros qui correspondent à une entrée de la *user file descriptor table* du processus.
- Un processus commence avec 3 entrées non vides
 - ▶ entrée n°0: descripteur de l'entrée standard (**stdin**)
 - ▶ entrée n°1: descripteur de la sortie standard (**stdout**)
 - ▶ entrée n°2: descripteur de la sortie erreur (**stderr**)
 - ▶ L'utilisation de redirections (<,>,>>) va éventuellement modifier les entrées concernées par ces redirections.
- À chaque appel à **fopen()**
 - ▶ On crée une nouvelle entrée dans la *user file descriptor table*
 - ▶ Cette entrée servira à stocker l'adresse du BCF associé au fichier logique nouvellement ouvert.

SYR1-L3 Info & Miage

111

4. Exemple de MEO des BCF : Unix



- Ici, les processus *i* et *j* travaillent sur le même fichier **f1.txt**.
- Ils disposent de leur propre BCF, mais partagent la même copie mémoire du descripteur de fichier (*inode*)

SYR1-L3 Info & Miage

112