

## Systèmes de gestion des fichiers

## Plan du chapitre

- Introduction
- Organisation logique
  - Fichiers
  - Désignation et catalogues
- Mise en œuvre des SGF
  - Gestion de l'espace libre
  - Descripteurs de fichiers
  - Amélioration des performances
- Fiabilité du SGF
- Le SGF de PedagOS

## Définitions

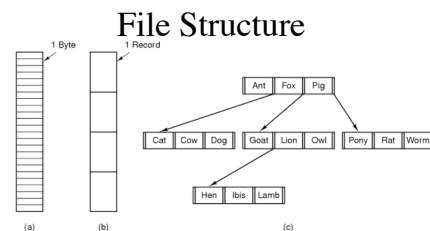
- Fichier = unité d'information
- Noms de fichiers et catalogues
- Système de gestion des fichiers : SGF

## Pourquoi des fichiers

- Permanence des informations
- Stockage d'informations de grande taille
- Partage entre utilisateurs

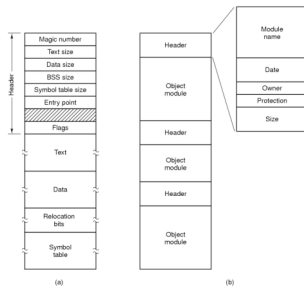
## Organisation logique : les fichiers

- Structure des fichiers
- Types, suffixes, attributs
- Accès aux fichiers
- Opérations sur les fichiers



- Three kinds of files
  - byte sequence
  - record sequence
  - tree

## File Types

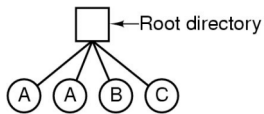


(a) An executable file (b) An archive

## Désignation des fichiers : catalogues

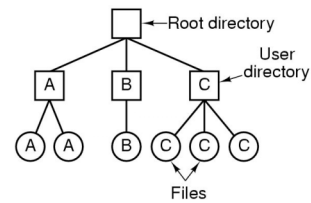
- Principe : conserver la correspondance entre un nom de fichier et sa localisation
- Dans une table appelée **catalogue** (directory)
- Catalogue unique, catalogue par utilisateur

## Directories Single-Level Directory Systems



- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C

## Two-level Directory Systems

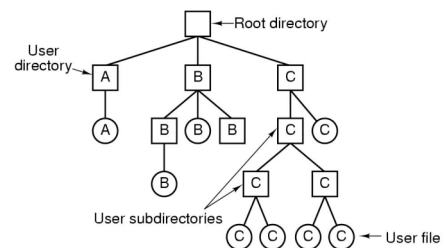


Letters indicate *owners* of the directories and files

## Désignation des fichiers : catalogues

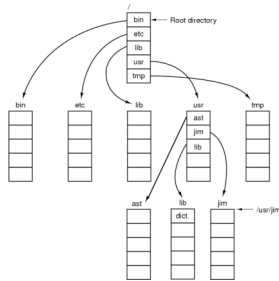
- Catalogues hiérarchiques
  - Désignation absolue, relative
  - Catalogue courant
  - . Et ..

## Hierarchical Directory Systems



A hierarchical directory system

## Path Names

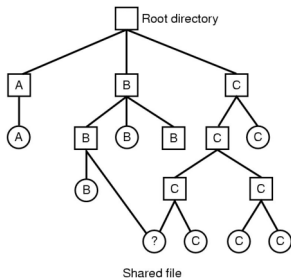


A UNIX directory tree

## Désignation des fichiers : catalogues

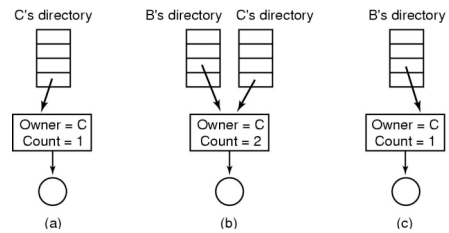
- Liens et partage de fichiers

## Shared Files (1)



File system containing a shared file

## Shared Files (2)



(a) Situation prior to linking

(b) After the link is created

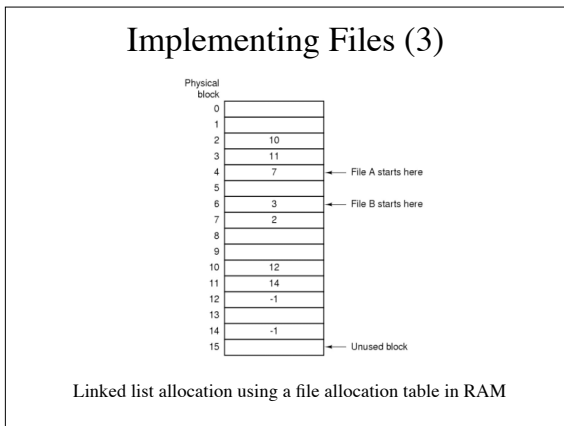
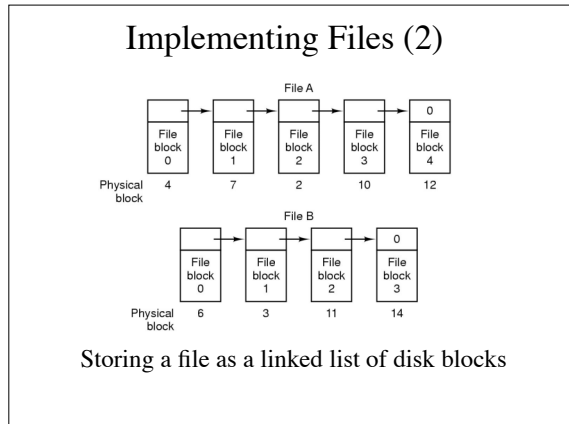
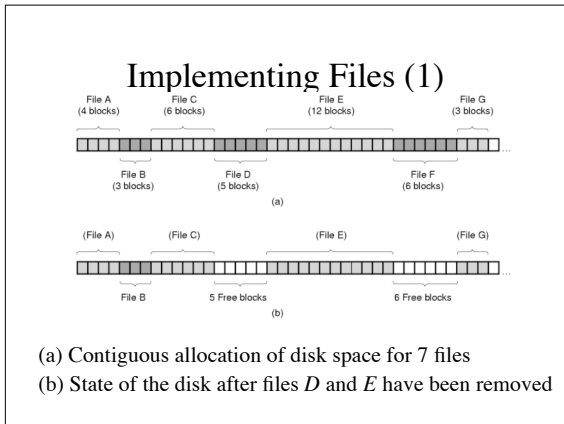
(c) After the original owner removes the file

## Mise en Œuvre du SGF

- Réalisation de base
- Extensions pour améliorer les performances

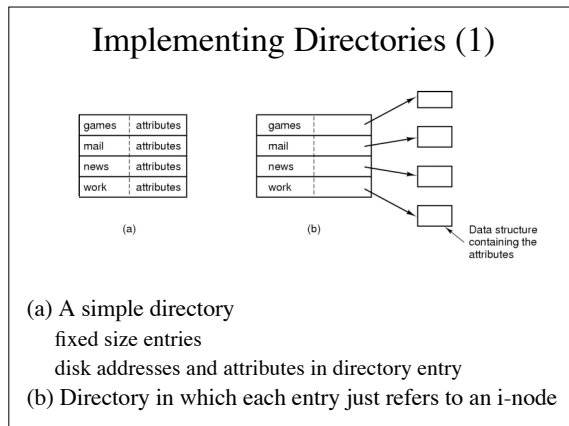
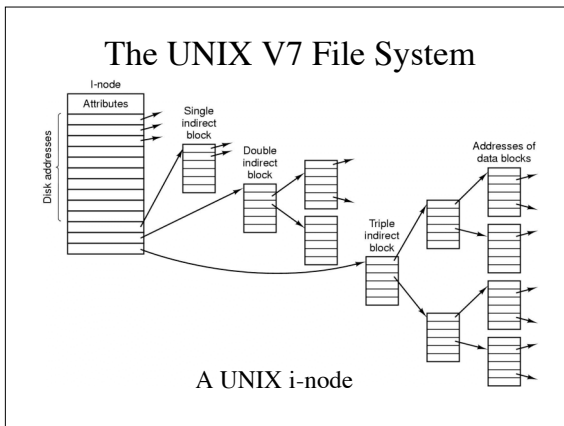
## Allocation d'espace disque

- Contraintes : espace de grande taille, accès à forte latence
- Allocation contiguë
- Allocation par blocs de taille fixe
- Gestion des blocs libres ou occupés, descripteurs



### Représentation des fichiers et des catalogues

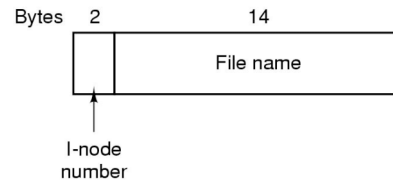
- Un exemple de descripteur : le I-node d'Unix
- Structure des catalogues



## Rappels

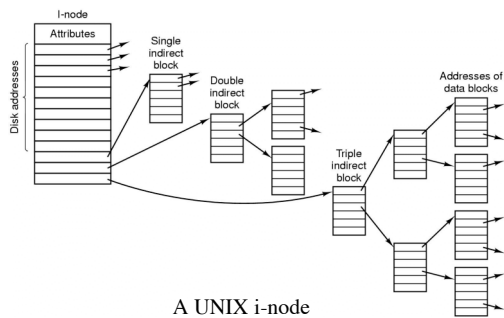
- Implantation d'un fichier décrite dans un descripteur, l'i-node du fichier
- Les catalogues sont des fichiers spéciaux
- On trouve à des emplacements connus :
  - La carte des blocs libres
  - L'i-node du catalogue racine

## The UNIX V7 File System (1)



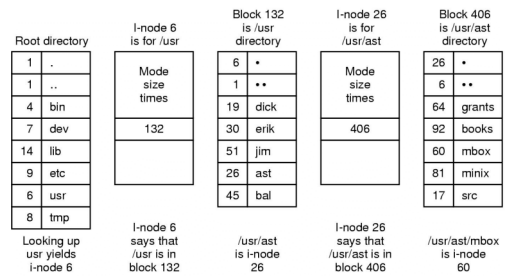
A UNIX V7 directory entry

## The UNIX V7 File System (2)



A UNIX i-node

## The UNIX V7 File System (3)



The steps in looking up /usr/ast/mbox

## Résumé

- Allocation d'espace disque par blocs de taille fixe
  - Comment choisir la taille ?
- Fichier = descripteur + ensemble de blocs
- Catalogue assure une correspondance entre une chaîne de caractères et un descripteur

## Le programme de la séance

- Comment améliorer l'efficacité
- Comment assurer la sécurité
  - Pannes
  - Malveillances
- Étude de cas : un SGF pour PedagOS

## Amélioration de l'efficacité

- Lenteur relative des disques, forte latence
- Accès séquentiel domine
- Plus de lectures que d'écriture
- Beaucoup de petits fichiers à faible durée de vie

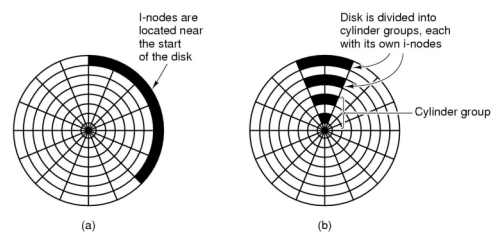
## Utilisation de tampons en mémoire : le cache du disque

- Transfert systématique d'un bloc entier en mémoire
- Recopie sur disque
  - soit à la fermeture,
  - soit pour libérer un bloc (SOS fiabilité)
  - soit écriture immédiate
- Algorithme de remplacement de bloc

## Autres améliorations

- Idée : limiter les déplacements du bras
- Partition en cylindres
  - Nombre fixe d'I-nodes
  - Bit maps pour emplacements libres
- Allocation des blocs pour favoriser l'accès séquentiel

## File System Performance



- I-nodes placed at the start of the disk
- Disk divided into cylinder groups
  - each with its own blocks and i-nodes

## Fichiers journalisés

- « Log structured file systems »
- La question des lectures est « résolue » via les tampons
- Il faut améliorer les performances des écritures
- Solution de principe : faire de grandes écritures séquentielles

## Fichiers journalisés (suite)

- La place des blocs et des i-nodes varie à chaque écriture
- Une table des i-nodes permet l'accès aléatoire
- Récupération de l'espace libre : réorganisation de tout le disque

## Comparaison

- Performances au niveau des meilleurs systèmes classiques en conditions « dures »
  - Tampons petits
  - Disques occupés à 80 %
- Sont meilleurs dans les autres cas
- Facilitent la récupération après panne

## Fiabilité du SGF

- Sauvegardes
- Reconstruction, cohérence des informations
- Restrictions d'accès
- Sécurité

## Les sauvegardes

- Idée : recopier les fichiers sur un autre support jugé plus sûr
- Disque miroir, disques raid
- Sur un support « permanent » (CD)
- Sauvegardes
  - Périodiques
  - Incrémentales
  - Journalisées

## Redondance des informations

- Principe : les informations sur disque sont redondantes et permettent de reconstruire des tables détruites
- Reconstruction de la carte des blocs libres
- Reconstruction des catalogues et des i-nodes

## Restrictions d'accès

- Utilisateurs et groupes
- Listes de contrôle d'accès
- Verrouillage
- Contrôle « lecteurs-rédacteurs »

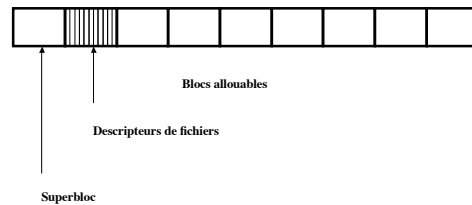
## Sécurité

- Ordinateur isolé : pas de grave problème
- Ordinateur connecté à un réseau
  - Authentification des utilisateurs
  - Les mots de passe
  - Carte à puce

## Les fichiers de PedagoS

- Structures sur disque
- Structures de données en mémoire centrale
- Programmation d'appels systèmes
  - fixer position
  - lecture
  - ouverture

## Organisation du disque : descripteurs et blocs



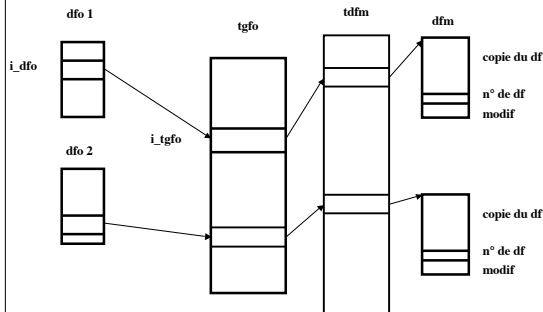
## Descripteurs de fichiers

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

## Accès au disque : cache

- Table gérée en adressage dispersé pour la recherche rapide
- Liste LRU pour le remplacement
- Utilisation via les procédures obtenir\_bloc et liberer\_bloc

## Structures en mémoire centrale



## Descripteur de fichier en mémoire

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



## Table des fichiers ouverts

- Conserve le mode d'ouverture et la position de chaque fichier ouvert

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

## Descripteur local à un processus

- Permet une désignation interne à un processus
- Utilisée pour les redirections des E/S standards

## Appel système fixer\_position

```
exec_fixer_position(i_dfo,position) {
    if ((i_dfo < 0) || (i_dfo >= MAX_FO_PROC))
        return (-1) ; // mauvaise valeur d'i_dfo
    if ( proelu->dfo [i_dfo] < 0) return (-1) ; // fichier non ouvert
    i_tgfo = proelu->dfo [i_dfo] ; // indice du fichier dans la tgfo
    dfm = tdfm [tgfo[i_tgfo]->L_tdfm ] ; // pointeur vers le dfm
    if ((position > dfm->taille_log) || (position < 0))
        return (-3) ; // nouvelle position invalide
    tgfo[i_tgfo]->position = position ; // nouvelle position dans la tgfo
    return (0) ; // appel système réussi
}
```

## Appel système de lecture

```
exec_lire(i_dfo,adr_mem,nbcar) {
    vérifier paramètres
    récupérer dfm
    position = (ibloc,depl)
    adr_disque = logique_physique (dfm,ibloc)
    obtenir_bloc(adr-disque)
    copie du bloc vers zone utilisateur
    libérer_bloc }
```

## Appel système d'ouverture

- Entrée : nom, mode
- Sortie : numéro de dfo
- Effet de bord : mise à jour des tables de fichiers ouverts et de descripteurs en mémoire

## Recherche dans les catalogues

Nom de fichier aaa/bbb/ccc

Numéro i\_df de descripteur de fichier  
(catalogue de recherche)

Recherche de aaa ds le catalogue i\_df, si cette recherche réussit, on obtient un nouvel i\_df dans lequel on va rechercher bbb

A la fin du nom, on a le descripteur recherché

## Ouverture (sans détection d'erreurs)

```
exec_ouvrir (nom,mode) {
    indice_df =recherche_arbre(nom,i_df) ;
    //on va maintenant construire les différentes tables décrivant le fichier
    i_tgfo = allouer_tgfo() ; // recherche emplacement dans tgfo
    i_dfo = allouer_dfo(proelu->dfo) ;
    proelu->dfo[i_dfo] = i_tgfo ;
    tgfo[i_tgfo]->cpt = 1 ;
    tgfo[i_tgfo]->mode = mode ;
    tgfo[i_tgfo]->position = 0 ; // début du fichier
    i_tdfm = obtenir_dfm(indice_df) ; //construction du dfm
    ++tdfm[i_tdfm]->cpt ; // augmentation compteur d'utilisation
    tgfo[i_tgfo]->i_tdfm = i_tdfm ;
    return(i_dfo) ;
}
```

## Programmation du cache

- Version 1 : un bloc est écrit sur disque à la récupération (cache « paresseux »)
- Version 2 : vidages anticipés

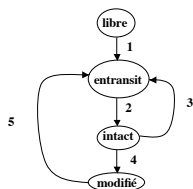
## Recherche d'un bloc

- Grand nombre de blocs
  - Recherche séquentielle inadaptée
- Utilisation d'une table auxiliaire
  - Adressage dispersé (« hash code »)

## Descripteur de bloc en mémoire

```
struct dbm {
    cpt ; //nombre d'utilisateurs du bloc
    ad_disque ; // adresse du bloc sur disque
    ad_mem ; // adresse du bloc en mémoire centré
    etat ; // défini plus loin
    liens ; // chaînage dans une liste LRU
}
```

## Cache paresseux - états des blocs



- 1 allocation d'un bloc libre
- 2 fin de chargement d'un bloc
- 3 réallocation d'un bloc intact
- 4 écriture dans un bloc intact
- 5 réallocation d'un bloc modifié

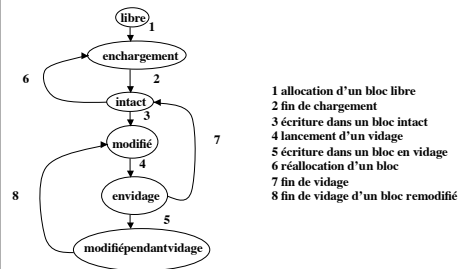
## Obtenir\_bloc (bloc en cache)

```
obtenir_bloc (bd) {
    dbm = recherche_en_cache (bd) ;
    if (dbm != NULL) { // le bloc est dans le cache
        //ou en cours de transfert
        if (dbm->etat == entransit) {
            /* le processus est bloqué et son pid placé
            dans la liste des processus en attente du bloc */
            dbm->cpt++ ;
            lancer_processus_suivant ;
        } ;
        dbm->cpt ++ ; chaîner dbm en tête de liste LRU ;
    }
}
```

## Obtenir\_bloc (bloc absent)

```
dbm = trouve_place_cache ;
if (dbm->etat != libre) oter_htable(dbm->ad_disque) ; /* on retire le bloc
if (dbm->etat == modifié) { vidage = vrai ;
copier (dbm, dbm_aux) ; ;
dbm->cpt = 1 ; //mise à jour dbm pour le nouveau bloc
dbm->etat = entransit ;
dbm->ad_disque = bd ;
ajouter_htable (bd,dbm) ;
chaîner dbm en tête de liste LRU ;
if (vidage) lancer_vidage (dbm_aux); // les opérations de vidage
lancer_chargement (dbm) ;// et chargement sont bloquantes
dbm->etat = intact ;
rendre éligibles les processus en attente du bloc ;
}
}
```

## Cache avec anticipation



## Vidage anticipé

- Vidage demandé par un processus spécialisé
- Le processus « bouche trou » peut être utilisé
- Si le système est chargé, ce n'est pas suffisant
- Le processus videur est activé périodiquement