## Fichiers

- **Définitions**
  - Fichier : ensemble d'informations regroupées en vue de leur conservation et de leur utilisation dans un système informatique
  - Système de gestion de fichiers (SGF) : partie du système d'exploitation qui conserve les fichiers et permet d'y accéder
- **Fonctions d'un SGF**
  - Conservation permanente des fichiers (permanente : indépendamment de l'exécution des programmes et de l'intégrité de la mémoire principale) -> conservation en mémoire secondaire (disque)
  - Organisation logique et désignation des fichiers
  - Partage et protection des fichiers
  - Réalisation des fonctions d'accès aux fichiers
- **Plan d'étude** (illustré par le SGF d'Unix)
  - Aspects logiques
    - Désignation
    - Fonctions d'accès aux fichiers
  - Notions sur la réalisation
  - Protection



## Place du SGF dans un système d'exploitation

- **Les fichiers jouent un rôle central dans un système d'exploitation**
  - Support des programmes exécutables
  - Support des données
  - Communication entre processus et entre utilisateurs
  - Lien étroit entre fichiers et entrées-sorties

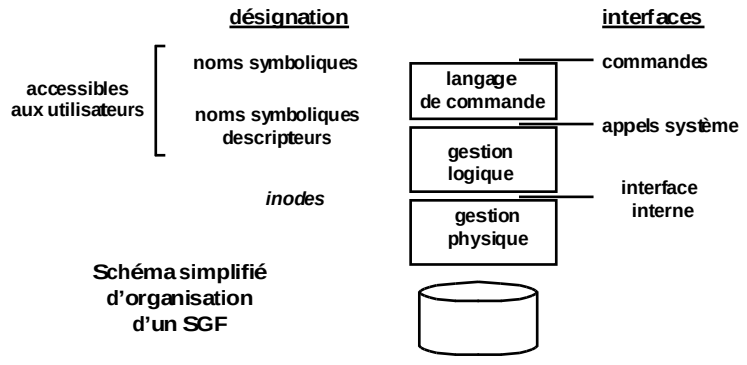


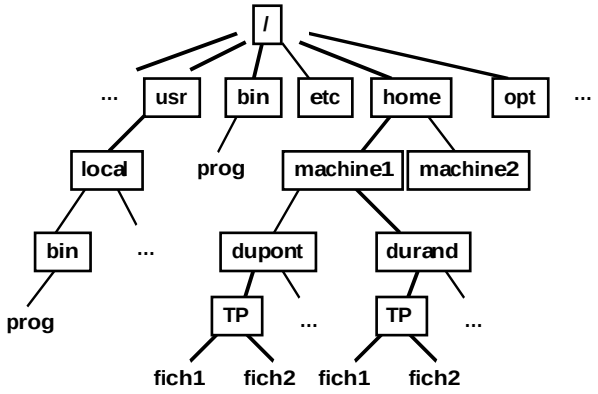

Schéma simplifié d'organisation d'un SGF

## Désignation des fichiers (1)

- **Principe de la désignation symbolique : organisation hiérarchique**
  - Les nomsforment une arborescence
  - Nœuds intermédiaires = catalogues - en anglais *directory* (ce sont aussi des fichiers)
  - Nœuds terminaux = fichiers simples
  - Nom (universel ou absolu) d'un fichier = le chemin d'accès depuis la racine (en anglais : *path*)

Exemples de noms universels :

/
/bin
/usr/local/bin/prog
/home/machine1/dupont/TP/fich1
/home/machine1/durand/TP/fich1






## Désignation des fichiers (2)

- **Divers raccourcis simplifient la désignation**
  - noms relatifs au catalogue courant

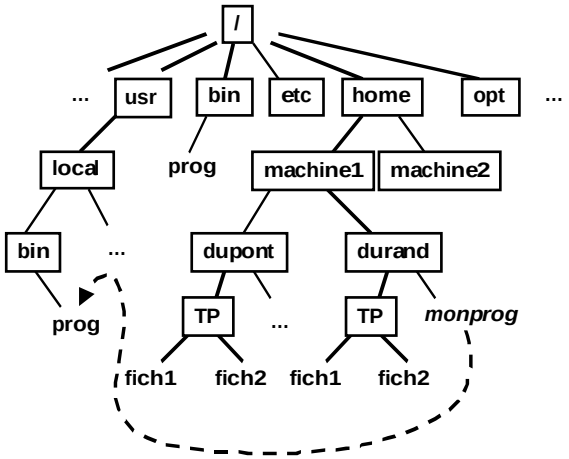
Si catalogue courant = /home/machine1/dupont/
alors on peut utiliser les noms relatifs
TP/fich1, TP/fich2

  - désignation du père : ..

Si catalogue courant = /home/machine1/dupont/
alors on peut utiliser
../durand/TP/fich1

  - liens symboliques

Si catalogue courant = /home/machine1/durand/




- création du lien : ln -s monprog /usr/local/bin/prog
- dans le catalogue courant, le nom monprog désigne maintenant le fichier /usr/local/bin/prog
- un lien n'est qu'un raccourci : si le fichier cible est supprimé, le lien devient invalide

## Désignation des fichiers (3)

- **Catalogue courant**
  - par défaut, tout usager a un catalogue courant de base (*home directory*), par exemple /home/machine/dupont ; un raccourci est ~dupont
  - on peut changer de catalogue courant au moyen de la commande
    cd <nom du nouveau catalogue> (*cd* tout seul ramène au catalogue de base)
  - le nom . désigne le catalogue courant
  - on peut connaître le nom absolu du catalogue courant par *pwd*
  - on peut connaître le contenu du catalogue courant par *ls* (*ls -l* est plus complet)
- **Un point fin : règles de recherche**
  - Pour exécuter un programme (fichier exécutable), il suffit d'entrer une commande avec son nom simple. Comment le système va-t-il trouver le fichier correspondant ?
  - Il existe une règle de recherche qui donne la suite de catalogues qui seront successivement explorés, dans cet ordre, pour trouver le fichier. Cette suite est enregistrée dans une variable d'environnement (PATH) définie au départ pour chaque usager, et que l'on peut modifier (a priori vous n'aurez pas à le faire)
  - Pour les programmes usuels (commandes du système, etc), la commande *which* indique le nom absolu du fichier qui sera exécuté par défaut
    Exemple :
    ```
    which gcc
    /usr/local/bin/gcc
    ```



## Utilisations courantes des fichiers

**Dans Unix, le contenu d'un fichier est simplement une suite de caractères, sans autre structure. L'interprétation de ce contenu dépend de l'utilisation**

- **Programmes exécutables**
  - Commandes du système ou programmes créés par un usager
  - Exemples
    ```
    gcc -o prog prog.c      produit le programme exécutable dans un fichier prog
    ./prog                  exécute le programme prog
    ```
    On aurait pu aussi écrire prog (sans ./). Quel est l'intérêt d'écrire ./prog ?
- **Fichiers de données**
  - Documents, images, programmes sources, etc.
  - Convention : il est commode de mettre au nom un suffixe indiquant la nature du contenu
    Exemples : .c (programme C), .o (binaire translatable, cf plus loin), .h (inclusions), .gif (un format d'images), .ps (PostScript), .pdf (Portable Document Format), etc.
- **Fichiers temporaires servant pour la communication**
  - Ne pas oublier de les supprimer après usage
  - On peut aussi utiliser des tubes (cf plus loin)

## Utilisation des fichiers dans le langage de commande

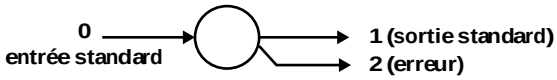
- **Créer un fichier**
  - Le plus souvent, les fichiers sont créés par les applications, non directement dans le langage de commande. Exemple : éditeur de texte, compilateur, etc
  - On peut néanmoins créer explicitement un fichier (cf plus loin, avec flots)
- **Créer un catalogue**
  - mkdir <nom du catalogue> ; le catalogue est initialement vide
- **Détruire un fichier**
  - rm <nom du fichier> ; il est recommandé de faire rm -i (demande de confirmation)
- **Détruire un catalogue**
  - rmdir <nom du catalogue> ; le catalogue doit être vide
- **Conventions pour les noms de fichiers**
  - * désigne n'importe quelle chaîne de caractères :
    rm *.o : détruit tous les fichiers dont le nom finit par .o

    ls *.c : donne la liste de tous les fichiers dont le nom finit par .c



## Fichiers et flots d'entrée-sortie

- **Il y a un lien étroit entre fichiers et entrées-sorties**
  - Les organes d'entrée-sortie sont représentés par des fichiers particuliers (dans le catalogue /dev)
  - Tout processus utilise des flots d'entrée-sortie qui peuvent être dirigés soit vers un fichier, soit vers un organe d'entrée-sortie

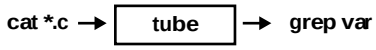



  - Par défaut : 0 vient du clavier, 1 et 2 vont vers l'écran
  - Un flot d'entrée peut être réorienté par <, un flot de sortie par >. Exemple
    cat fich — affiche à l'écran le contenu du fichier fich
    cat fich > fich1 — crée le fichier fich1 (s'il n'existe pas) et y met le contenu de fich
    cat /dev/null > fich — crée un fichier (vide) de nom fich
  - La lecture ou l'écriture d'un fichier sont des opérations séquentielles (suite d'octets à partir d'un pointeur courant) - cf plus loin

## Tubes

- Un tube (*pipe*) est un fichier qui sert de tampon entre deux processus fonctionnant en producteur-consommateur
  - Exemple (langage de commande)

    cat *.c | grep var
  - Créé deux processus : p1 exécute cat *.c, p2 exécute grep var
  - Connecte le flot de sortie de p1 à l'entrée d'un tube, le flot d'entrée de p2 à la sortie du tube

    
    
  - Question : effet de cette commande ?
- On peut combiner plusieurs tubes entre eux, et avec des flots d'entrée-sortie
  - Exemple

    cat f1 f2 f3 | grep toto | wc -l > result

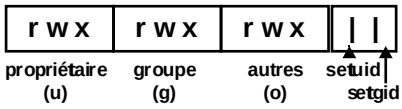


## Sécurité des fichiers (1)

- Définition (générale) de la sécurité
  - confidentialité : informations accessibles aux seuls usagers autorisés
  - intégrité : pas de modifications non désirées
  - contrôle d'accès : seuls certains usagers sont autorisé à faire certaines opérations
  - authentification : garantie qu'un usager est bien celui qu'il prétend être
- Comment assurer la sécurité
  - Définition d'un ensemble de règles (politiques de sécurité) spécifiant la sécurité d'une organisation ou d'une installation informatique
  - Mise en place de mécanismes (mécanismes de protection) pour assurer que ces règles sont respectées
- Sécurité des fichiers (dans Unix)
  - On définit
    - des types d'opérations sur les fichiers : lire, écrire, exécuter (correspondent respectivement à des contraintes de confidentialité, intégrité, contrôle d'accès)
    - des classes d'usagers
      - usager propriétaire du fichier
      - groupe propriétaire
      - tous les autres

## Protection des fichiers (2)

- **Fichiers ordinaires**




exemple (fichier fich) : rwx r-- r-- : tout accès pour le propriétaire, lecture seule pour tous les autres

chmod go+w fich : donne le droit w au groupe et aux autres
chmod o-w fich : retire le droit w aux autres

  - r = lecture, w = écriture, x = exécution
- **Catalogues**
  - même chose, mais le droit x signifie "recherche dans le catalogue"
- **Un mécanisme de délégation**
  - Le problème : partager un programme dont l'exécution nécessite des droits d'accès que n'ont pas les usagers potentiels
  - Solution (*setuid* ou *setgid*) : pour l'exécution de ce programme (et uniquement pour cette exécution), un usager quelconque reçoit temporairement les droits de l'usager ou ou du groupe propriétaire
- **Règles d'éthique**
  - protéger vos informations confidentielles
  - ne pas tenter de contourner les mécanismes de protection
  - les règles de bon usage s'appliquent indépendamment de la protection (ce n'est pas parce qu'un fichier n'est pas protégé qu'il est licite de le lire)

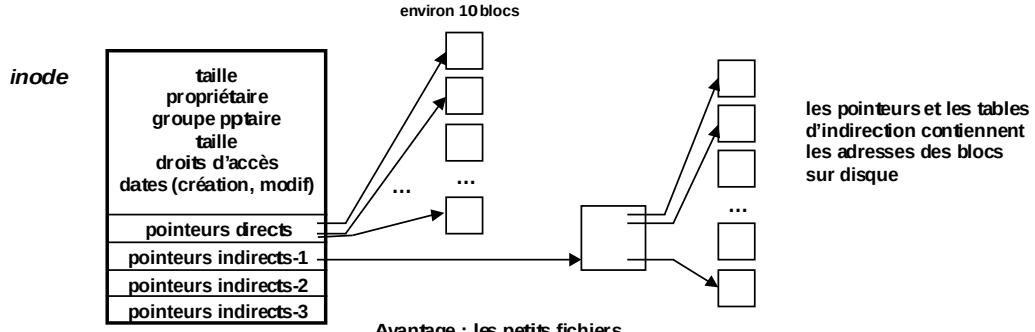


## Interface système pour l'utilisation des fichiers

- **Appels système**
  - primitives d'accès : *open* (ouverture ou création), *close*, *read*, *write*, *unlink*
  - au niveau de l'interface système, un fichier est désigné par un descripteur (ou poignée, en anglais *handle*)
  - la lecture et l'écriture sont séquentielles (à partir de la position courante d'un pointeur, initialement au début du fichier) ; le pointeur peut être déplacé par *lseek*
- **Exemples simples**
  - ouvrir un fichier en lecture seule (déclarer son intention de l'utiliser)
    fd = open ("/home/machine/toto/fich", O_RDONLY)
    le résultat est un descripteur du fichier (ou -1 si ouverture impossible)
  - lire dans le fichier que l'on vient d'ouvrir
    n = read (fd, buf, 20)
    lit 20 octets dans le tampon buf, supposé réservé. n indique le nombre d'octets lus, qui peut être inférieur à 20 si le fichier contient moins de 20 caractères
  - fermer le fichier
    close(fd)
    le descripteur fd n'est plus relié à aucun fichier ; il est inutilisable
  - d'autres exemples seront vus en TP

## Réalisation des fichiers dans Unix (1)

- **Représentation physique d'un fichier**
  - Un fichier est représenté physiquement par un ensemble de blocs (suite d'octets de taille fixe) sur disque
  - Typiquement, la taille d'un bloc est 8K (peut varier selon réalisations)
- **Structure de données pour la gestion interne des fichiers**
  - La structure principale (invisible aux usagers) associée à un fichier est un descripteur appelé *inode* du fichier

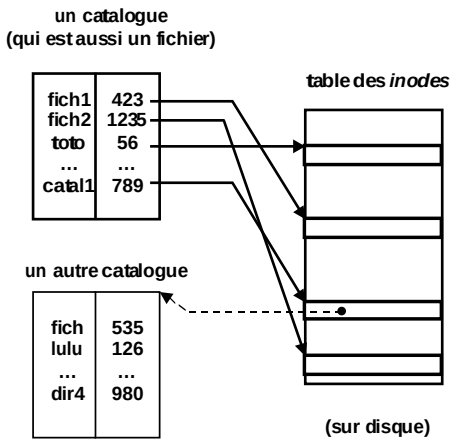



Avantage : les petits fichiers
(les plus nombreux) sont représentés de manière efficace



## Réalisation des fichiers dans Unix (2)

- **Correspondance entre noms symboliques et représentation physique**
  - Les *inodes* sont conservés dans une table, et chaque entrée d'un catalogue pointe vers l'*inode* correspondant




- **Avantages**
  - les *inodes* ne contiennent pas le nom des fichiers, ce qui simplifie les modifications (changement de nom, etc)
  - pour améliorer les performances, les *inodes* et les catalogues sont conservés en cache (en mémoire principale)

## Quelques statistiques d'usage sur les fichiers

- **Taille**
  - La plupart des fichiers sont de petite taille (taille médiane : quelques Koctets) ; mais il y a aussi de très grands fichiers (quelques Goctets), en raison du développement des applications multimédia
- **Durée de vie**
  - Beaucoup de fichiers ont une durée de vie très brève (quelques secondes) ; ce sont les fichiers temporaires utilisés pour les échanges
  - Quand un fichier survit à la phase initiale, il dure généralement très longtemps
- **Accès**
  - Une forte majorité des accès (entre 2/3 et 3/4) sont des lectures
  - La plupart des accès sont séquentiels, et concernent l'ensemble du fichier
  - Les accès possèdent la propriété de localité (accès récents = bonne estimation des accès futurs)
- **Partage**
  - Le partage simultané des fichiers est rare
- **Intérêt de ces statistiques**
  - pour améliorer la conception des applications
  - pour améliorer la conception des SGF



## Résumé de la séance 4

- **Notion de fichier**
  - conservation permanente de l'information
  - partage de l'information
  - désignation (espace de noms)
- **Le SGF d'Unix**
  - désignation symbolique : arborescence de fichiers, noms absolus, relatifs, liens
  - les opérations primitives : *open*, *close*, *read*, *write*
  - fichiers, flots de données, "tubes"
  - quelques idées sur la réalisation
- **Protection**