Systèmes et Réseaux – MIAGE 2
Systèmes d'Exploitation

Fichiers

Sara Bouchenak

Sara.Bouchenak@imag.fr
http://sardes.inrialpes.fr/~bouchena/teaching

Rôle des fichiers

- Les fichiers jouent un rôle important dans un système informatique
  - Support de programmes exécutables
  - Support de données
  - Communication entre processus
  - Communication entre utilisateurs
  - Lien étroit entre fichiers et entrées-sorties

Définitions

- Fichier
  - Ensemble d'informations regroupées en vue de leur conservation et de leur utilisation dans un système informatique
- Système de gestion de fichier (SGF)
  - Sous-système d'un système d'exploitation
  - Rôle
    - Conservation permanente (sur disque) des fichiers
    - Organisation logique et désignation des fichiers
    - Partage et protection des fichiers
    - Réalisation des fonctions d'accès aux fichiers

Plan

1. Introduction aux systèmes d'exploitation
2. Processus
3. Gestion des processus

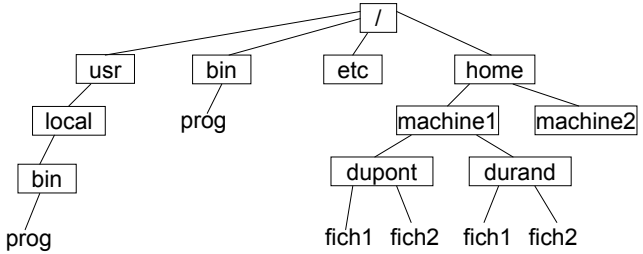
4. ***Fichiers***
   - ***Désignation***
   - ***Fonctions d'accès aux fichiers***
   - ***Flots, entrées-sorties, tubes***
   - ***Protection***
   - ***Notions de réalisation***

5. Mémoire
6. Etude de cas

Désignation des fichiers

- Principe de la désignation symbolique
  - Organisation hiérarchique en arbre
  - Nœuds intermédiaires de l'arbre : répertoires (*directories*)
  - Nœuds terminaux de l'arbre : fichiers simples (*files*)
  - Nom absolu d'un fichier : chemin d'accès depuis la racine de l'arbre (*path*)

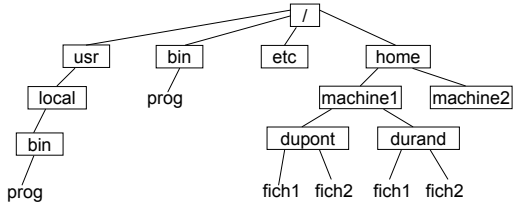
Désignation des fichiers (2)



- Exemples de noms absolus :
  /
  /bin
  /usr/local/bin/prog
  /home/machine1/dupont/fich1
  /home/machine1/durand/fich1

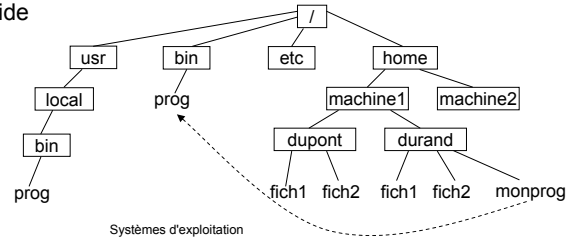
Désignation des fichiers (3)

- Divers raccourcis simplifient la désignation
  - Nom relatif au répertoire courant
    - Si le répertoire courant = /home/machine1
      alors on peut utiliser les noms relatifs
      dupont/fich1, dupont/fich2
  - Désignation du père
    - Si le répertoire courant = /home/machine1/dupont
      alors on peut utiliser ../durand/fich1



Désignation des fichiers (4)

- Autres raccourcis
  - Liens symboliques
    - Si répertoire courant = /home/macine1/durand
    - Création du lien :
      ln –s /usr/bin/prog monprog
    - Dans le répertoire 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 (5)

- Répertoire courant
  - Répertoire dans lequel on est situé
  - Commande *pwd* : pour connaître le nom absolu du répertoire courant
- Répertoire de base
  - Par défaut, tout utilisateur a un répertoire de base (*home directory*)
  - Exemple :
    /home/machine/dupont pour l'utilisateur dupont,
    un raccourci est *~dupont*

Désignation des fichiers (6)

- Contenu d'un répertoire
  - Commande *ls*
  - Exemple 1 :
    <unix> ls /home
    machine1 machine2
    <unix>
  - Exemple 2 : Si répertoire courant est /home/machine1/dupont
    <unix> ls
    fich1 fich2
    <unix>
- Changement de répertoire courant
  - Commande *cd <nom du nouveau répertoire>*
  - *cd* sans paramètre ramène au répertoire de base

Règles de recherche

- Les noms des commandes tapés dans un shell sont des raccourcis
  Exemple : *ls*, *gcc*
- Ces raccourcis permettent d'éviter de taper le noms absolu du fichier exécutable (programme) de la commande
  Exemple : */bin/ls*, */usr/local/bin/gcc*
- Comment est-ce que le shell retrouve le fichier exécutable correspondant à une commande-raccourcie ?

Règles de recherche (2)

- Variable d'environnement PATH
  - Commande *echo* : valeur de la variable PATH
    <unix> echo $PATH
    /usr/local/bin:/usr/bin:/usr/j2se/bin
    <unix>
  - Répertoires de le variable PATH successivement explorés pour rechercher un fichier portant le nom de la commande-raccourcie tapée
  - Ordre des répertoires dans le PATH important

Règles de recherche (3)

- Commande *which*
  - Indique le nom absolu du fichier qui sera exécuté par cette commande
  - Exemple :
    <unix> which gcc
    /usr/local/bin/gcc
    <unix>
  - Comment fonctionne la commande *which* ?
- Vous avez écrit un programme qui s'appelle *gcc* et vous voulez l'exécuter (et pas le *gcc* correspondant à */usr/local/bin/gcc*).
  Deux solutions ; lesquelles ?
  - Exécuter *./gcc* (dans le répertoire où se trouve votre programme)
  - Redéfinir la variable d'environnement PATH

Variables d'environnement

- Commande *env* : affichage de l'ensemble des variables d'environnement définies dans un shell et de leurs valeurs
- Commande *setenv* :
  - Définition d'une nouvelle variable d'environnement, exemple : setenv CLASSPATH .:./classes
  - Modification de la valeur d'une variable existante, exemple : setenv PATH .:$PATH
- Commande *unsetenv* : élimination d'une variable d'environnement

Utilisations courantes des fichiers

- Programmes exécutables
  - Commande du système ou commande créée par un utilisateur
  - Exemple : Produire un programme exécutable dans un fichier
    <unix> gcc –o prog prog.c
  - Exemple : Exécuter un fichier exécutable (programme)
    <unix> ./prog

Utilisations courantes des fichiers (2)

- Fichiers de données
  - Exemples :
    - Programmes sources
    - Images
    - Audio/vidéo
  - Convention : suffixe à la fin du nom de fichier, selon le type de données, exemples :
    - .c pour fichier de programmes sources C
    - .gif pour fichiers d'images
    - .ps pour fichier PostScript

Plan

1. Introduction aux systèmes d'exploitation
2. Processus
3. Gestion des processus
4. ***Fichiers***
   - ***Désignation***
   - ***Fonctions d'accès aux fichiers***
   - ***Flots, entrées-sorties, tubes***
   - ***Protection***
   - ***Notions de réalisation***
5. Mémoire
6. Etude de cas

Utilisation de fichiers dans le langage de commande (shell)

- Créer un fichier
  - Le plus souvent, les fichiers sont créés par des applications directement, non directement par le langage de commande
    Exemple : éditeur de texte
  - On peut néanmoins créer explicitement un fichier dans le langage de commande (c.f. entrées-sorties plus loin)
- Créer un répertoire
  - mkdir <nom du nouveau répertoire>
  - Le répertoire créé est initialement vide

Utilisation de fichiers dans le langage de commande (2)

- Détruire un fichier
  - rm <nom du fichier>
  - Il est recommandé d'utiliser la commande *rm –i* (demande de confirmation avant suppression effective)
- Détruire un répertoire
  - rmdir <nom du répertoire>
  - Le répertoire doit être initialement vide

Convention pour les noms de fichiers/répertoires

- * désigne n'importe quelle chaîne de caractères dans le nom d'un fichier ou répertoire
- Exemples d'utilisation
  - rm *.o :
    détruit tous les fichiers dont le nom se termine par .o
  - ls a*z :
    affiche tous les fichiers et répertoires dont le nom commence par *a* et se termine par *z*

Interface système pour l'utilisation de fichiers

- Descripteur de fichier
  - Au niveau de l'interface d'appels système, un fichier est représenté par un descripteur
  - Les descripteurs sont représentés par des entiers
- Ouverture d'un fichier
  - Avant d'utiliser un fichier, il faut l'ouvrir pour lui allouer un descripteur
  - Exemple :
    fd = open("/home/machine/durand/fich", O_RDONLY, 0)
    - Fichier ouvert en lecture seule (O_RDONLY), autres modes d'ouverture (O_WRONLY, O_RDWRITE)
    - Le fichier est crée s'il n'existe pas (voir également *creat*)
    - Le descripteur de fichier alloué est retourné (-1 si erreur)

Interface système pour l'utilisation de fichiers (2)

- Fermeture de fichier
  - Quand on a fini d'utiliser un fichier, il faut le fermer
  - Commande :
    close(fd)
  - Le descripteur *fd* n'est plus utilisable ; il pourra être réalloué par le système

Interface système pour l'utilisation de fichiers (3)

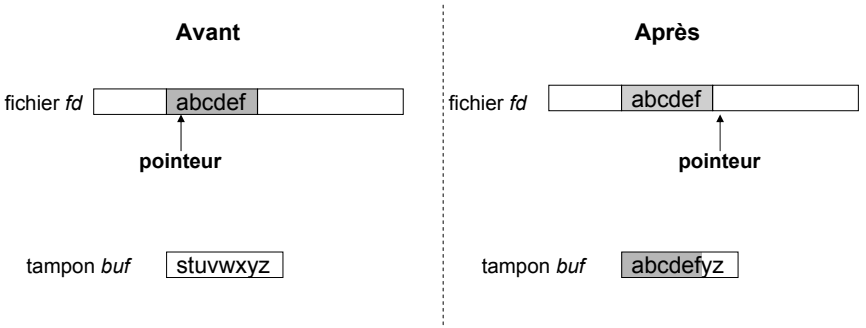
- Position courante dans un fichier
  - A l'ouverture du fichier, la position courant dans le fichier est initialisée à 0
  - Cette position est déplacée
    - Indirectement, par les opérations de lecture (*read*) et d'écriture (*write*), (c.f. détails plus loin)
    - Directement, par l'opération *lseek*

Position dans un fichier

- Exemple
  - Soient :
    - fd, un descripteur de fichier
    - sa position courante = 21
    - sa taille courante = 62
  - Cas 1 : lseek(fd, 30, SEEK_CUR)
    - +30 octets depuis la position courante
    - nouvelle position = 51, taille = 62
  - Cas 2 : lseek(fd, 71, SEEK_SET)
    - place le pointeur à la position 71
    - nouvelle position = 71, taille = 71

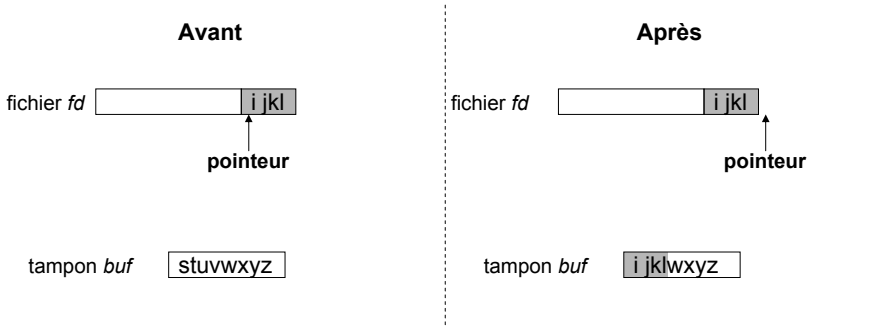
Interface système pour l'utilisation de fichiers : *read*

p = read(fd, buf, 6)



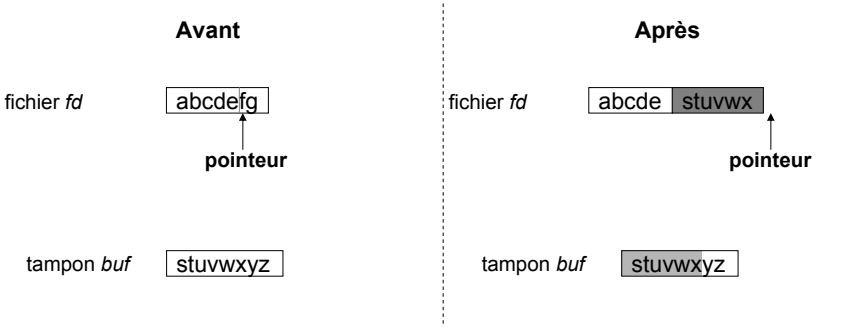
Interface système pour l'utilisation de fichiers : *read* (2)

p = read(fd, buf, 6)



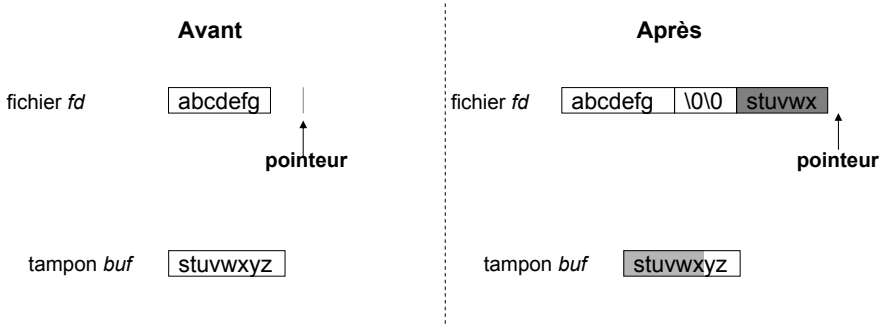
Interface système pour l'utilisation de fichiers : *write*

p = write(fd, buf, 6)



Interface système pour l'utilisation de fichiers : *write* (2)

p = write(fd, buf, 6)



Bibliothèques des fonctions d'accès aux fichiers

- Bibliothèque standard
  - Ensemble de fonctions de haut niveau d'accès aux fichiers
  - Exemples :
    fopen, fread, fwrite, fscanf, fprintf, fflush, fseek, fclose
- Bibliothèque *csapp* pour TP
  - Ensemble de fonctions avec gestion automatique des erreurs
  - Exemples :
    Open, Read, Write, Lseek, Close

Plan

1. Introduction aux systèmes d'exploitation
2. Processus
3. Gestion des processus
4. ***Fichiers***
   - ***Désignation***
   - ***Fonctions d'accès aux fichiers***
   - ***Flots, entrées-sorties, tubes***
   - ***Protection***
   - ***Notions de réalisation***
5. Mémoire
6. Etude de cas

Fichiers et flots d'entrées-sorties

- Lien étroit entre entrées-sorties et fichiers
  - Tout processus utilise des flots d'entrées-sorties
    - Entrée standard (clavier)
    - Sortie standard (écran)
    - Sortie d'erreur (écran)
  - Les flots d'entrées-sorties sont représentés par des fichiers (dans le répertoire /dev)
  - Les descripteurs de fichiers associés à ces flots sont, par convention
    - 0 pour l'entrée standard
    - 1 pour la sortie standard
    - 2 pour la sortie d'erreur
  - Les flots d'E/S peuvent être réorientés vers des fichiers

Manipuler les flots d'entrées-sorties : commandes

- Réorientation des flots standards
  - Au moyen de < et >
  - Exemples :
    - *cat fich* : écrit le contenu de *fich* sur la sortie standard (l'affiche à l'écran)
    - *cat fich* > *fich1* : réoriente la sortie standard de la commande *cat fich* dans le fichier *fich1* (copie *fich* dans *fich1*)

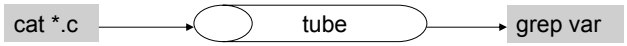
Tubes : Fichiers pour communication de processus

- Les tubes (*pipes*) permettent de faire communiquer des processus
- Un tube est un fichier qui sert de tampon (*buffer*) entre deux processus fonctionnant selon le modèle producteur-consommateur

- Exemple : cat *.c | grep var
  - Lister toutes les occurrences de la chaîne de caractère "var" dans les fichiers dont le nom se termine par *.c
  - La sortie (résultat) de *cat *.c* est réorientée vers l'entrée (second paramètre) de *grep var*

Tubes : Détails sur l'exemple

- cat *.c | grep var
  - Crée un processus p1 qui exécute la commande *cat *.c*
  - Crée un processus p2 qui exécute la commande *grep var*
  - Crée un tube entre ces deux processus
  - Connecte la sortie de p1 à l'entrée du tube
  - Connecte la sortie du tube à l'entrée de p

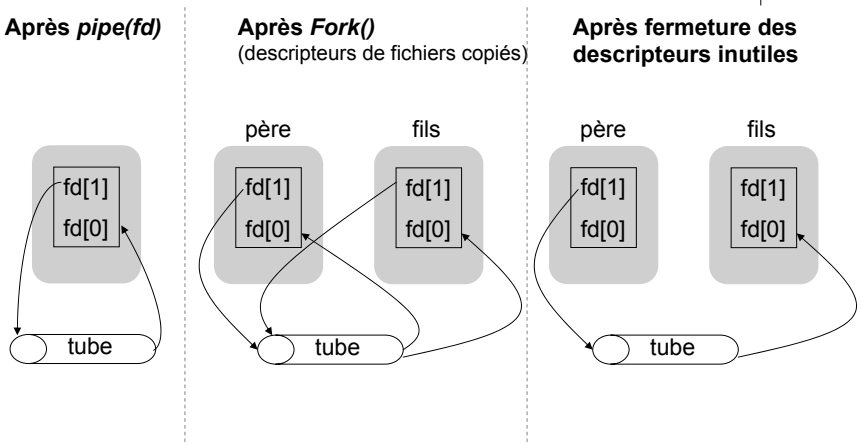


- Que fait la commande suivante ?
  - cat f1 f2 f3 | grep toto | wc –l > result

Manipuler les flots d'entrées-sorties : primitives

- Les tubes et les flots peuvent être manipulés par des primitives système
- Créer un tube
  - Primitive *pipe*
  - Exemple : int fd[2]; pipe(fd);
  - L'entrée et la sortie du tube sont associées à des descripteurs choisis par le système
  - Si la primitive réussit :
    - crée un tube
    - fd[1] : descripteur d'entrée du tube
    - fd[0] : descripteur de sortie du tube
    - 0 est retournée par *pipe*
  - Si la primitive échoue, elle renvoie -1

Tube : Communication d'un père avec son fils



Programmation d'un tube entre un père et un fils

testpipe.c

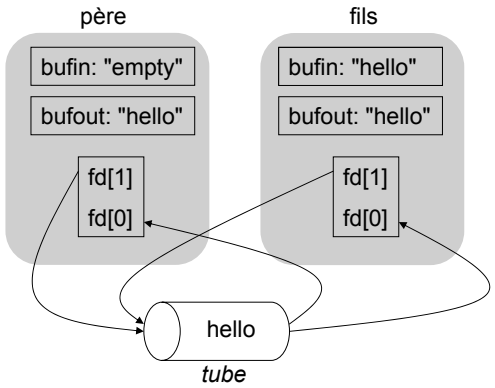
```
#include "csapp.h"
#define BUFSIZE 10
Int main(void) {
      char bufin[BUFSIZE] = "empty";          char bufout[ ] = "hello";
      int bytesin;          pid_t childpid;          int fd[2];

      Pipe(fd);
      bytesin = strlen(bufin);
      childpid = Fork();
      if (childpid != 0) {   /* père */
            Close([fd[0]);          Write(fd[1], bufout, strlen(bufout)+1);
            printf("[Parent process %d]: my bufin is {%s}, mybufout is {%s}\n", getpid(), bufin, bufout);

      } else {          /* fils */
            Close(fd[1]);          bytesin = Read(fd[0], bufin, BUFSIZE);
            printf("[Child process %d]: my bufin is {%s}, mybufout is {%s}\n", getpid(), bufin, bufout);
      }
      exit(0);
}
```


Programmation d'un tube entre un père et un fils (suite)

- Quelle est la trace d'exécution ?
  - Message affiché par le père
  - Message affiché par le fils

Programmation d'un tube entre un père et un fils : Exécution

Après Read(fd[0], bufin, …) chez le fils



Programmation d'un tube entre un père et un fils : Trace d'exécution

```
<unix> ./testpipe
[Parent process 29196]: my bufin is {empty}, my bufout is {hello}
[Child process 29197]: my bufin is {hello}, my bufout is {hello}
<unix>
```

FIFO : Tube nommé

- Communication entre un père et un fils
  - Un tube ne peut être utilisé qu'entre un processus et ses descendants
  - En effet, les extrémités (entrée et sortie) d'un tube ne sont désignées que par des descripteurs, qui ne peuvent se transmettre qu'entre père et fils
- Communication entre processus quelconques
  - Un **FIFO** est un tube spécial dit **tube nommé**
  - Un FIFO possède un nom symbolique tel qu'un fichier

FIFO : Manipulations

- Créer un FIFO
  - Primitive *mkfifo*
    #include <sys/types.h>
    #include <sys/stat.h>
    int mkfifo(char* nom, mode_t mode) renvoie 0 si ok, -1 si erreur
  - Cette primitive crée un FIFO appelé *nom*, avec le mode de protection *mode* (comme pour un fichier, c.f. détails protection plus loin)

FIFO : Manipulations

- Ouvrir un FIFO
  - Pour pouvoir être utilisé, un FIFO doit être préalablement ouvert par deux processus
  - L'un en mode écriture (entrée du FIFO)
  - L'autre en mode lecture (sortie du FIFO)
  - Chacun des processus reste bloqué tant que l'autre processus n'a pas ouvert le FIFO

Copie de descripteurs : *dup*

- Primitive *dup*
  - dup(fd)
  - Recopie le descripteur de numéro *fd* dans le premier descripteur disponible (descripteur disponible dans le système, de plus petit numéro)
- Primitive *dup2*
  - dup2(fd1, fd2)
  - Recopie le descripteur de numéro *fd1* dans le descripteur de numéro *fd2*
- Rôle de ces primitives
  - Rediriger les flots d'entrées-sorties

Exemple d'utilisation de *dup*

- Commande : *cat > toto*
  - Recopie ce qui est lu sur l'entrée standard (clavier) vers le fichier toto; l'entrée doit finir par EOF (control-D)
  - En fait, l'entrée standard est redirigée vers un fichier donné
- Fonctionnement
  - Commande *cat > toto* basée sur une duplication de descripteurs de fichiers associés à l'entrée standard du *cat* et au fichier *toto*

Plan

1. Introduction aux systèmes d'exploitation
2. Processus
3. Gestion des processus
4. ***Fichiers***
   - ***Désignation***
   - ***Fonctions d'accès aux fichiers***
   - ***Flots, entrées-sorties, tubes***
   - ***Protection***
   - ***Notions de réalisation***
5. Mémoire
6. Etude de cas

Sécurité : Rappels

- Définitions générales de la sécurité
  - Intégrité : pas de modifications non désirées des données
  - Confidentialité : informations accessibles aux seuls utilisateurs autorisés
  - Contrôle d'accès : seuls certains utilisateurs sont autorisés à faire certaines opérations
  - Authentification : garantir qu'un utilisateur est bien celui qu'il prétend être
- Comment assurer la sécurité
  - Définir un ensemble de règles (*politiques de sécurité*) spécifiant la sécurité d'une installation informatique
  - Mise en place de mécanismes (*mécanismes de protection*) pour assurer que ces règles sont respectées

Sécurité des fichiers (Unix)

- On définit
  - Des types d'opérations sur les fichiers
    - lire,écrire, exécuter
    - avec des contraintes de contrôle d'accès, de confidentialité et d'intégrité
  - Des classes d'utilisateurs
    - Utilisateur propriétaire du fichier
    - Groupe propriétaire
    - Tous les autres

Protection des fichiers

- Commande : ls –l
  - Affiche les droit d'accès aux fichiers et répertoires dans le répertoire courant
  - Exemple
    <unix> ls –l
    - rwx r - - r - - 1 dupont miage2 94 Oct 5 2005 fich1
    - rwx r - x r - - 1 dupont miage2 4320 Oct 5 2005 fich2
- Classes d'utilisateurs
  - u : utilisateur propriétaire
  - g : groupe propriétaire
  - o : tous les autres
- Types d'opérations
  - r : lecture
  - w :écriture
  - x : exécution

Protection des fichiers : exemples

- Exemple de fichier : *fich1*
  - rwx r - - r - -
  - Tout accès pour l'utilisateur propriétaire, accès en lecture pour le groupe propriétaire et pour les autres
- Exemple 1
  - Commande : chmod go+w fich1
  - Donne le droit *w* (écriture) au groupe et aux autres
- Exemple 2
  - Commande : chmod o-w fich1 ; que fait cette commande ?
  - Retire le droit *w* (écriture) aux autres

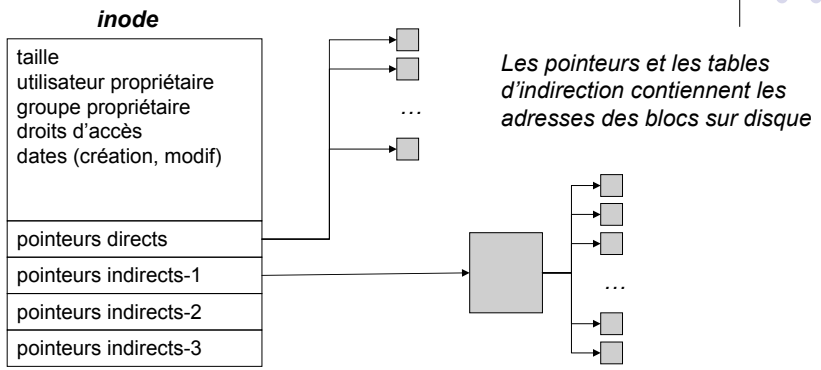
Plan

1. Introduction aux systèmes d'exploitation
2. Processus
3. Gestion des processus
4. ***Fichiers***
   - ***Désignation***
   - ***Fonctions d'accès aux fichiers***
   - ***Flots, entrées-sorties, tubes***
   - ***Protection***
   - ***Notions de réalisation***
5. Mémoire
6. Etude de cas

Réalisation des fichiers dans Unix

- 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 de 8Koctets (peut varier selon les réalisations)
- Structure de données pour la gestion interne des fichiers
  - La structure principale (invisible aux utilisateurs) associée à un fichier est un descripteur appelé *inode* du fichier
  - Les *inodes* sont contenus dans une table globale

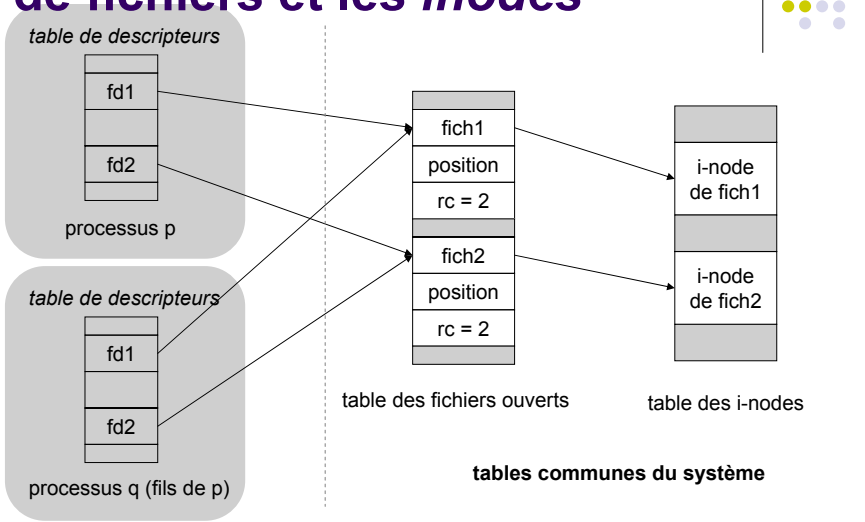
Réalisation des fichiers dans Unix : 2



Les pointeurs et les tables d'indirection contiennent les adresses des blocs sur disque

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

Relation entre les descripteurs de fichiers et les *inodes*



Quelques statistiques

- Taille
  - La plupart des fichiers sont de petite taille (quelques Koctets)
  - Mais il y a aussi de très grands fichiers (quelques Goctets), ex. documents multimédia
- Durée de vie
  - Beaucoup de fichiers ont une durée de vie très brève (quelques secondes), ex. fichiers temporaires utilisés pour les échanges
  - Quand un fichier survit à la phase initiale, il dure généralement très longtemps

Quelques statistiques (2)

- Accès
  - Une forte majorité des accès sont des lectures (entre 2/3 et 3/4)
  - La plupart des accès sont séquentiels et concernent l'ensemble du fichier
  - Les accès possèdent la propriété de localité : accès récents donnent une bonne estimation des accès futurs
- Partage
  - Le partage simultané des fichiers est rare
- Intérêt de ces statistiques
  - Améliorer la conception des applications
  - Améliorer la conception des SGF

Plan

1. Introduction aux systèmes d'exploitation
2. Processus
3. Gestion des processus
4. ***Fichiers***
   - ***Désignation***
   - ***Fonctions d'accès aux fichiers***
   - ***Flots, entrées-sorties, tubes***
   - ***Protection***
   - ***Notions de réalisation***
5. Mémoire
6. Etude de cas

Références

- **Systèmes d'exploitation – 2ème édition**, A. Tanenbaum, Pearson Education, 2003.
- **Practical UNIX Programming**, K. A. Robbins, S. Robbins, Prentice Hall, 1996
- **Principe des systèmes d'exploitation des ordinateurs**, S. Krakowiak, Dunod, 1985.
- Ce cours a été conçu à partir d'autres supports :
  - Sacha Krakowiak, http://sardes.inrialpes.fr/people/krakowia/
  - Fabienne Boyer, http://sardes.inrialpes.fr/people/boyer/cours/SR/