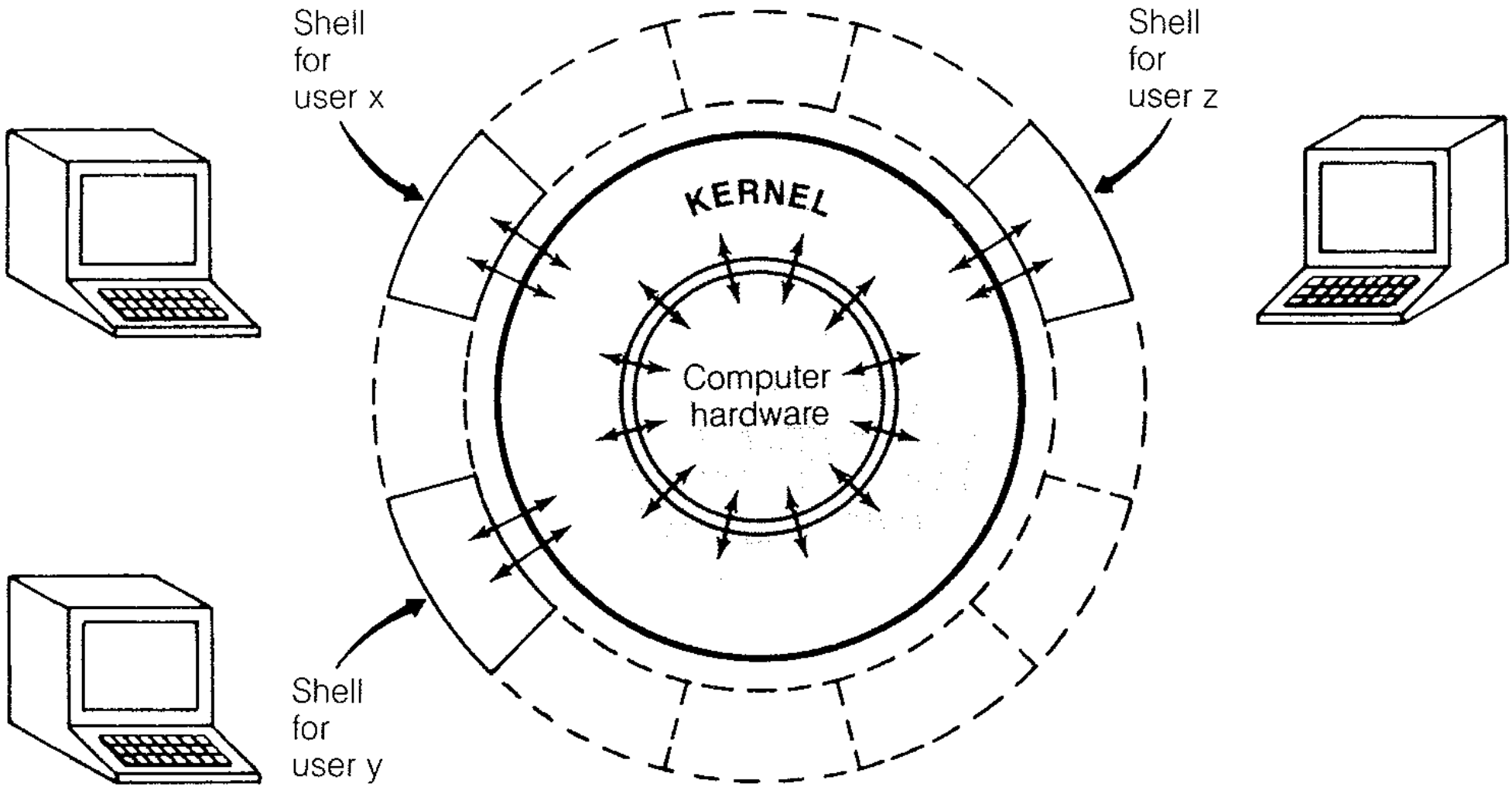
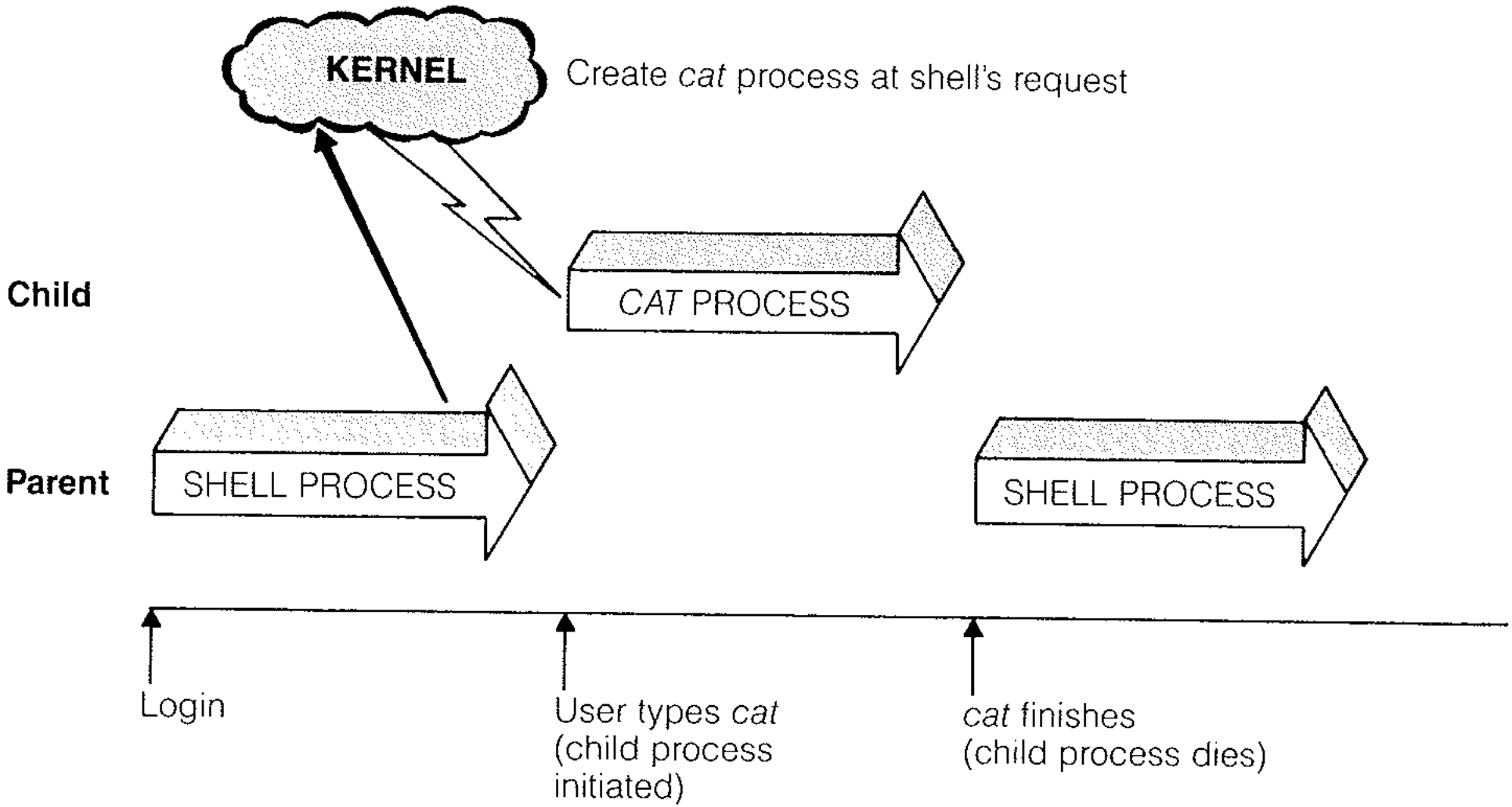
Shell Script

Georgios Arhodakis
Université Paris Dauphine

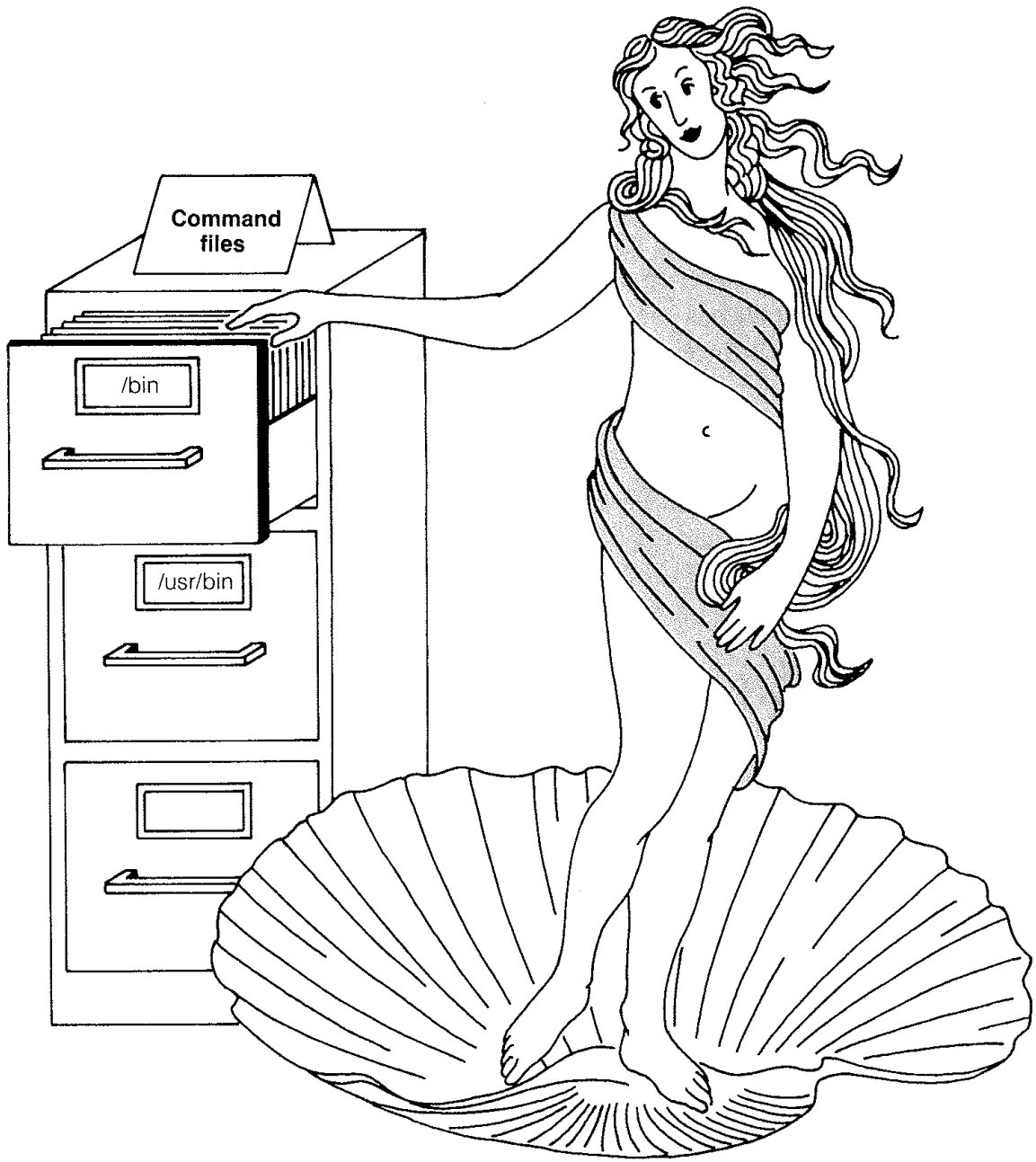
Le noyau et les « shells »



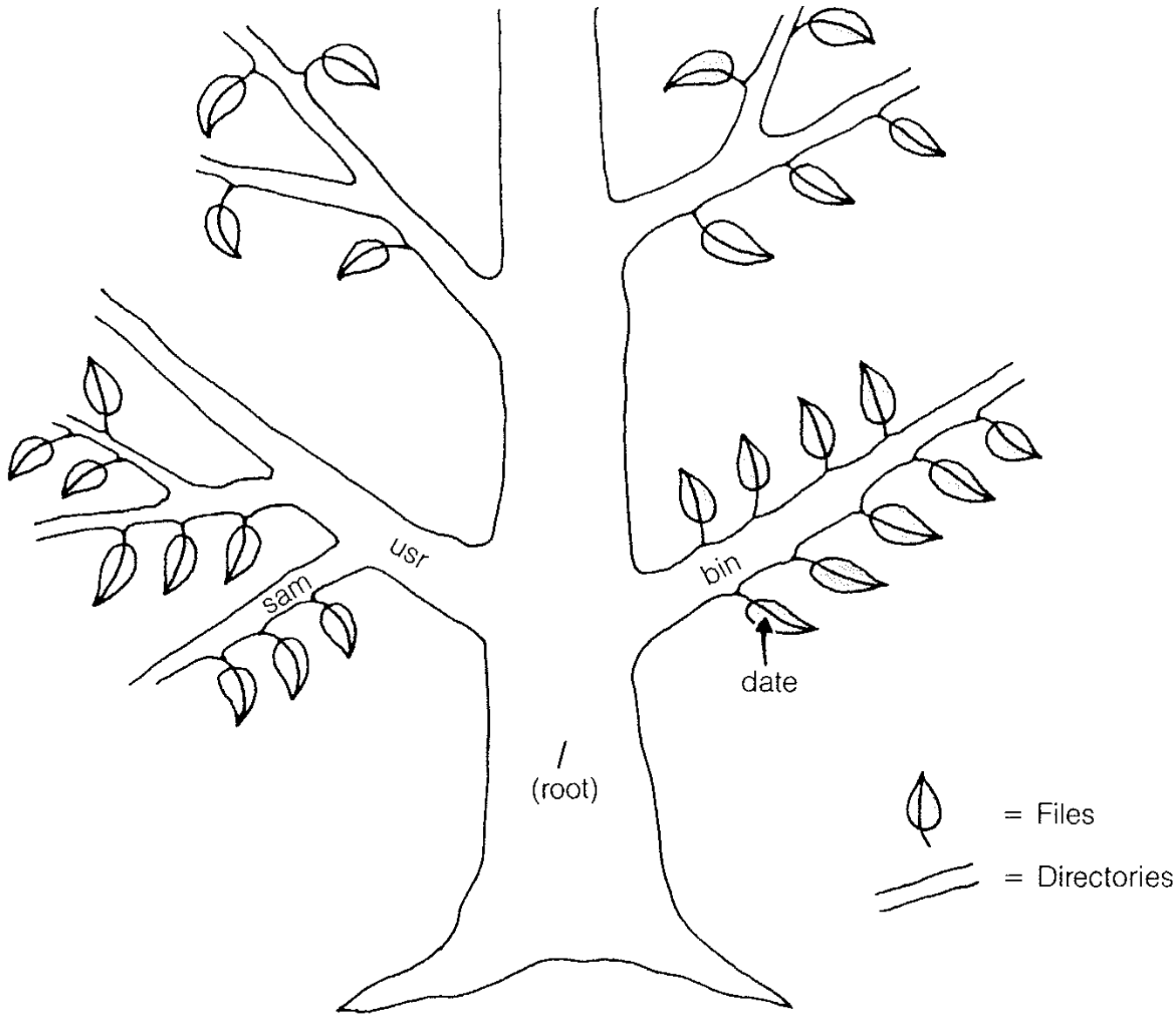
Processus « père & fils »



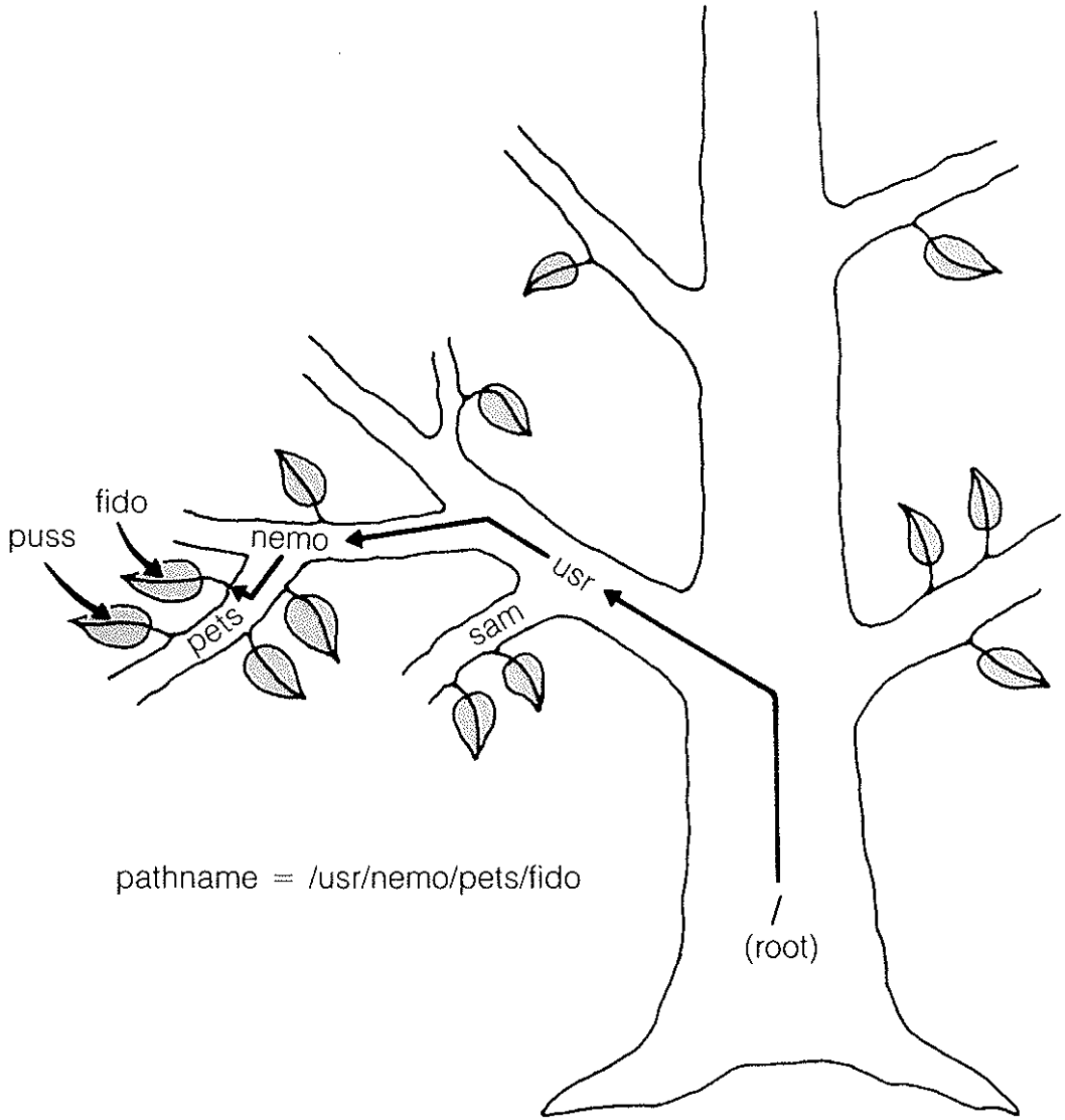
« fetch » la commande utilisateur



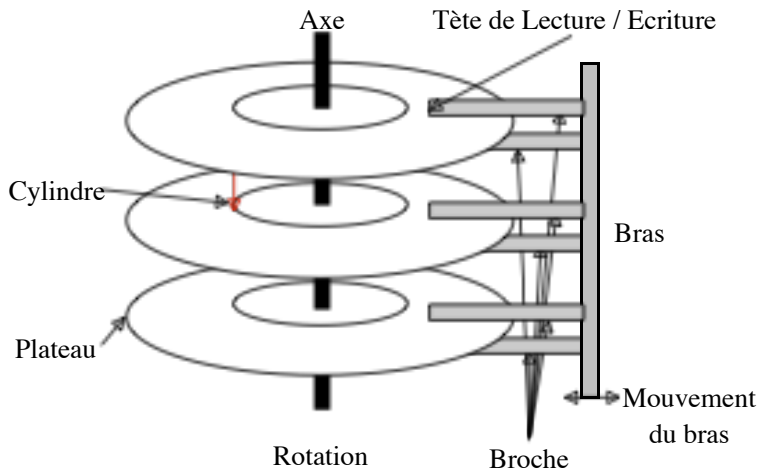
« Arborescence »

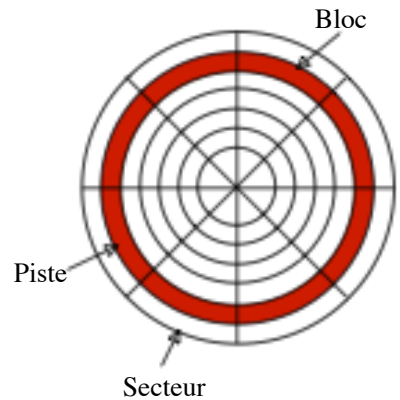


« chercher » dans le F.S. « fido »



Géométrie et Découpage d'un disque





Partitionnement d'un disque

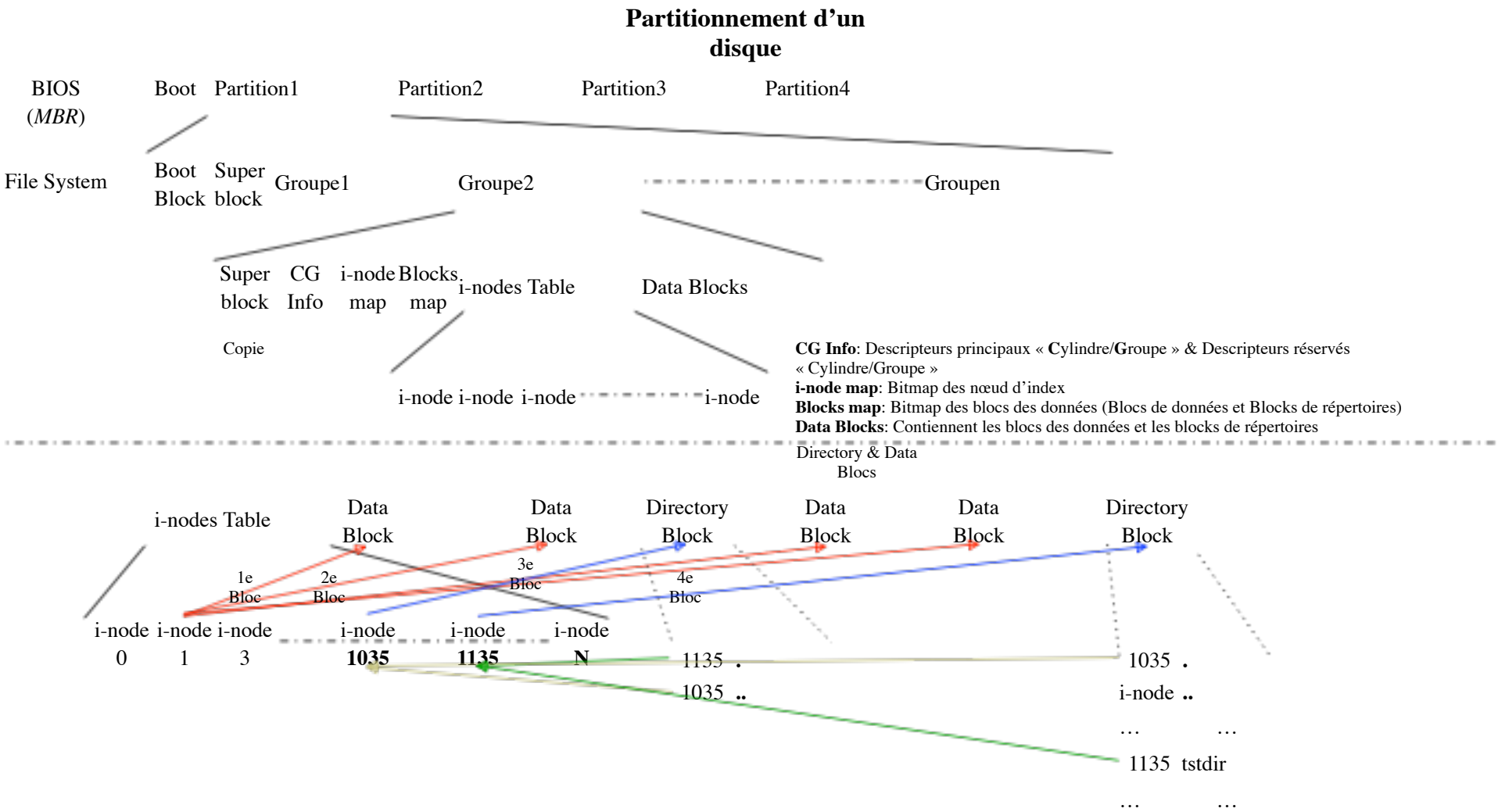
Partition étendue

BIOS (*MBR*)	Boot	Partition1	Partition2	Partition3	Partition4		
UEFI (*GPT*)	Boot	Entête primaire GPT	Table d'index GPT	GP1		GP128	Entête '*backup*' GPT

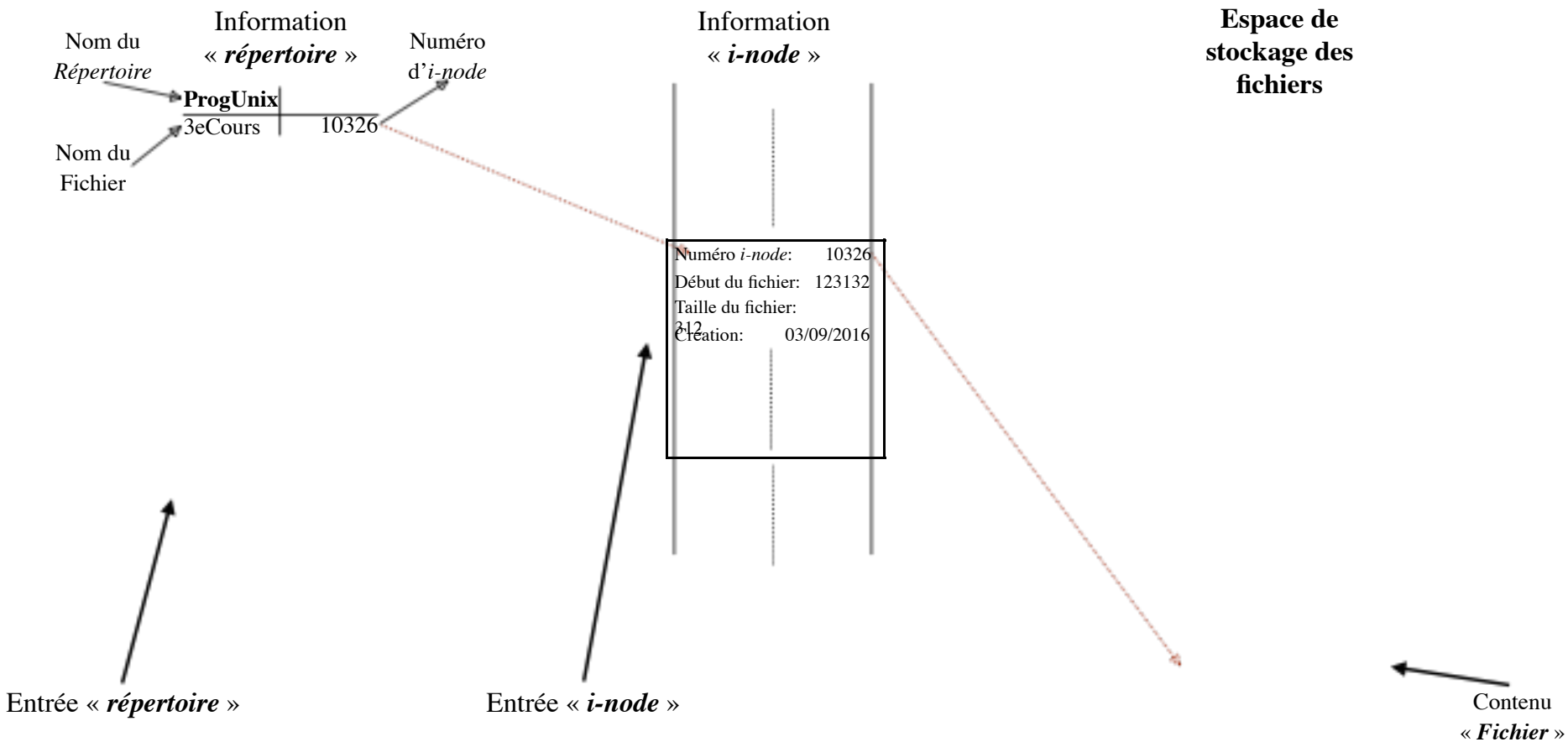
MBR ***protégé*** Sauvegarder en fin de disque

BIOS – Basic Input / Output System
MBR – Master Boot Record
UEFI – Unified Extensible Firmware Interface
GPT – GUID Partition Table
GUID – Globally Unique IDentifier
GP – GUID Partition

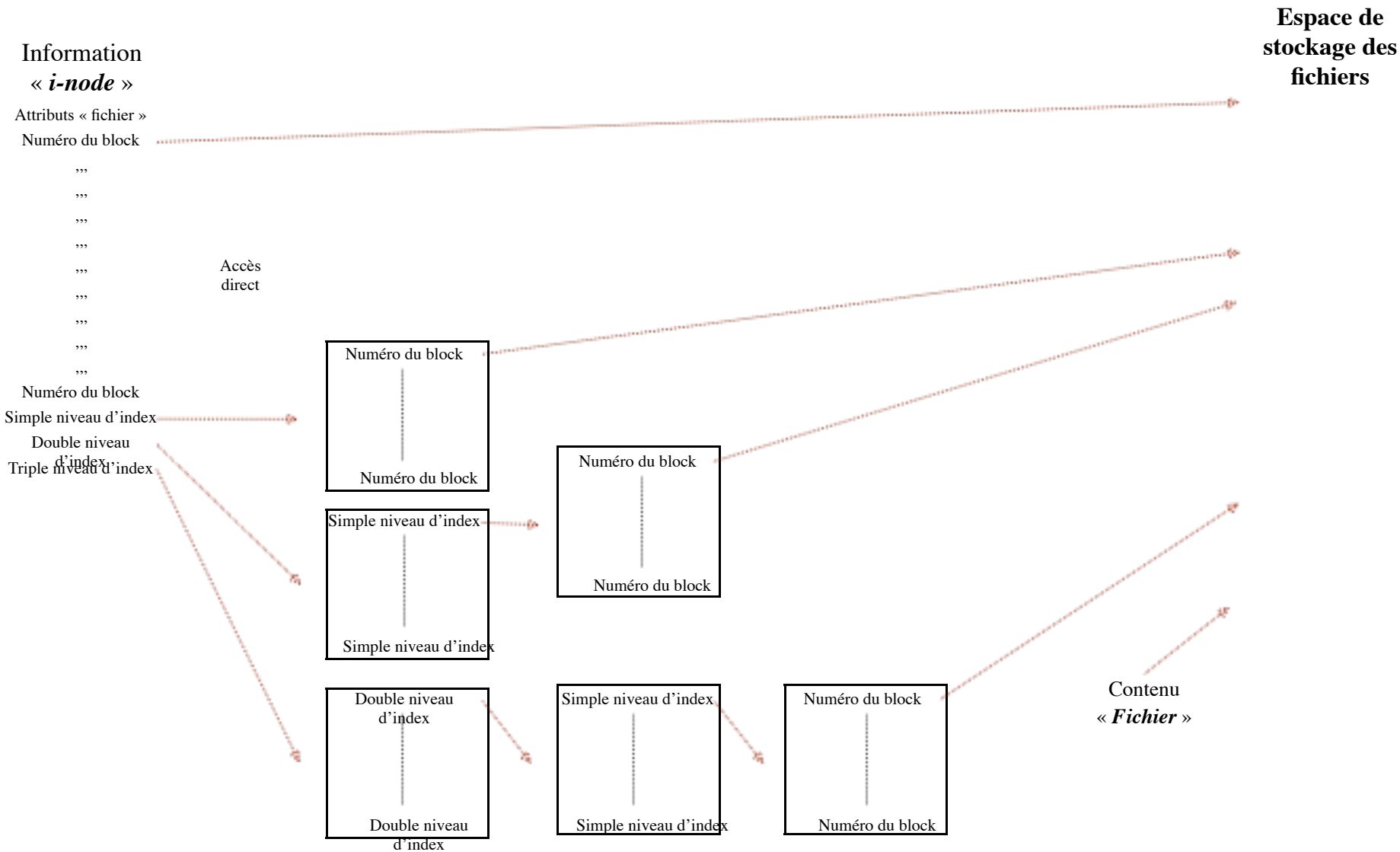
Système de fichiers



i-node (nœud d'index)



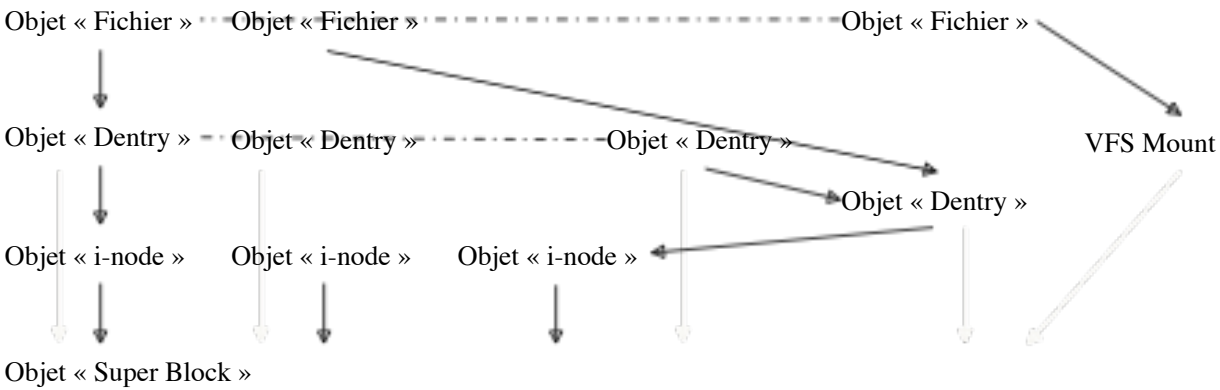
Allocation d'espace



Système de fichiers - VFS

VFS - <u>V</u>irtual <u>F</u>ile system
<u>S</u>witch

Système de fichiers - VFS



Relation entre « objets »

Applications

Espace Utilisateur

API – Application Program Interface

« Dentry » Cache

« i-node » Cache

Vue « évoluée »

FS0 FS1 FS2 FSi FSj FSo FSp FSy FSz

Espace Système

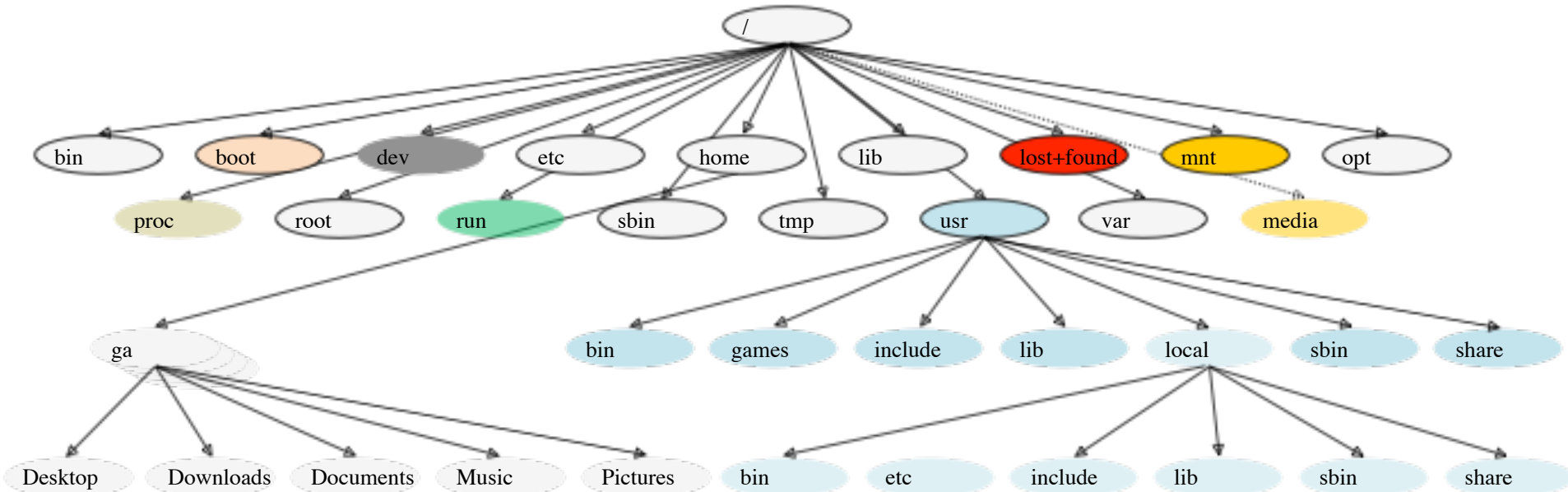
« Page » Cache

« Buffer » Cache

Pilotes de Périphériques

DENTRY – Directory Entry
FS – File **S**ystem

Système de fichiers



Structure arborescente

Au moins un système de fichier présent - le « ***root** F.S. (**F**ile **S**ystem) - /* »

Tout F.S., autre que le « / F.S. », peut être associé/dissocié du système selon les besoins utilisateur

Commandes:

mkfs – création d'un F.S. de type [ext2, ext3, ext4, msdos, …]

fsck – tester et 'éventuellement' réparer d'un F.S.

mount / **umount** – Associer/Dissocier un FS au « / » F.S.

df – Utilisation de l'espace [nb inodes, utilisé, libre, %, Associer à] d'un F.S.

Set bit

```
#ifdef    __STDC__
static    void s_bm(unsigned char *map, int c)
#else
static    void s_bm(map, c)
unsigned char *map;
int    c;
#endif
{
   map[c >> 3]  |= 1 << (c & 0x07);
   return;
}
```

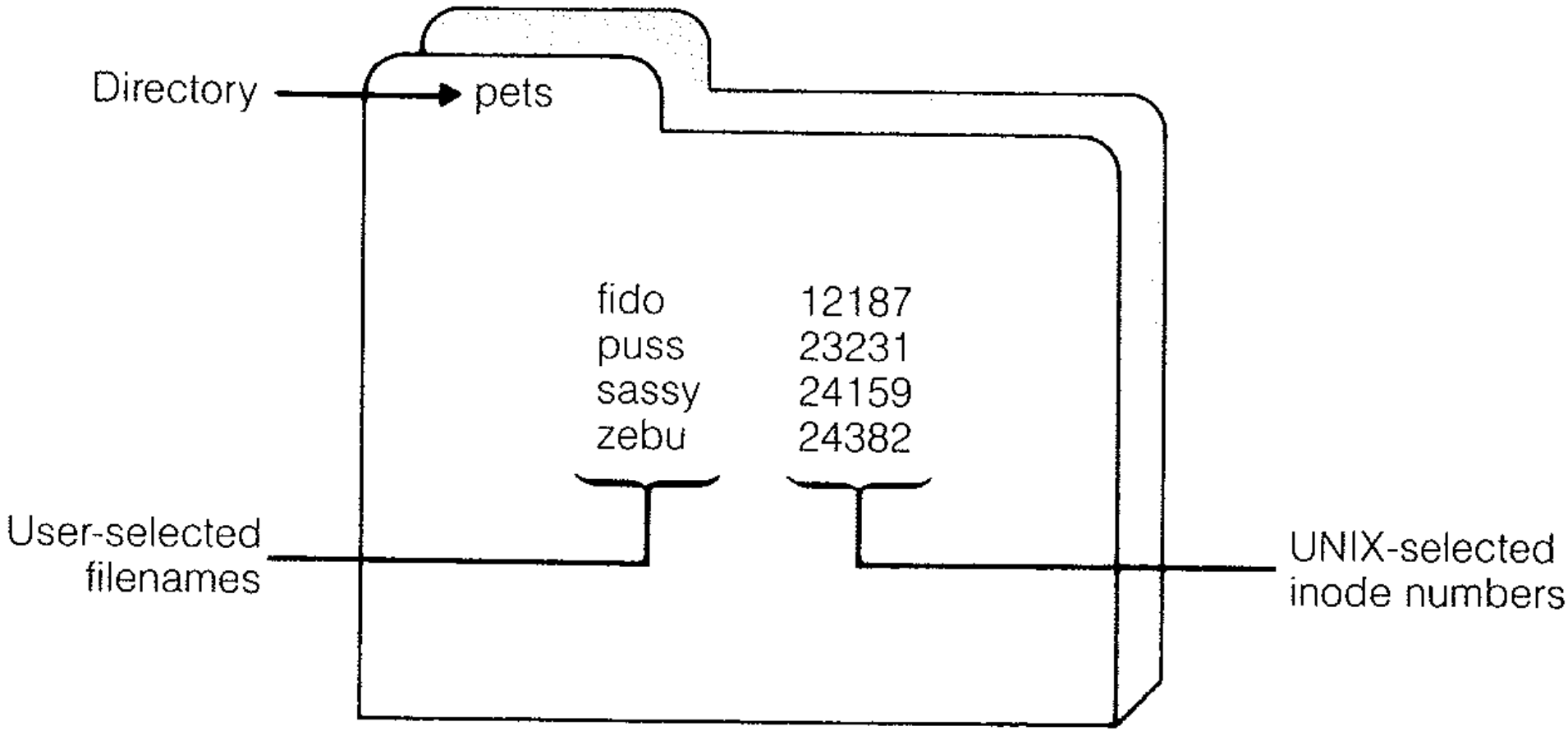
ReSet bit

```
#ifdef    __STDC__
static    void r_bm(unsigned char *map, int c)
#else
static    void r_bm(map, c)
unsigned char *map;
int    c;
#endif
{
   map[c >> 3]  &=  ~(1 << (c & 0x07));
   return;
}
```

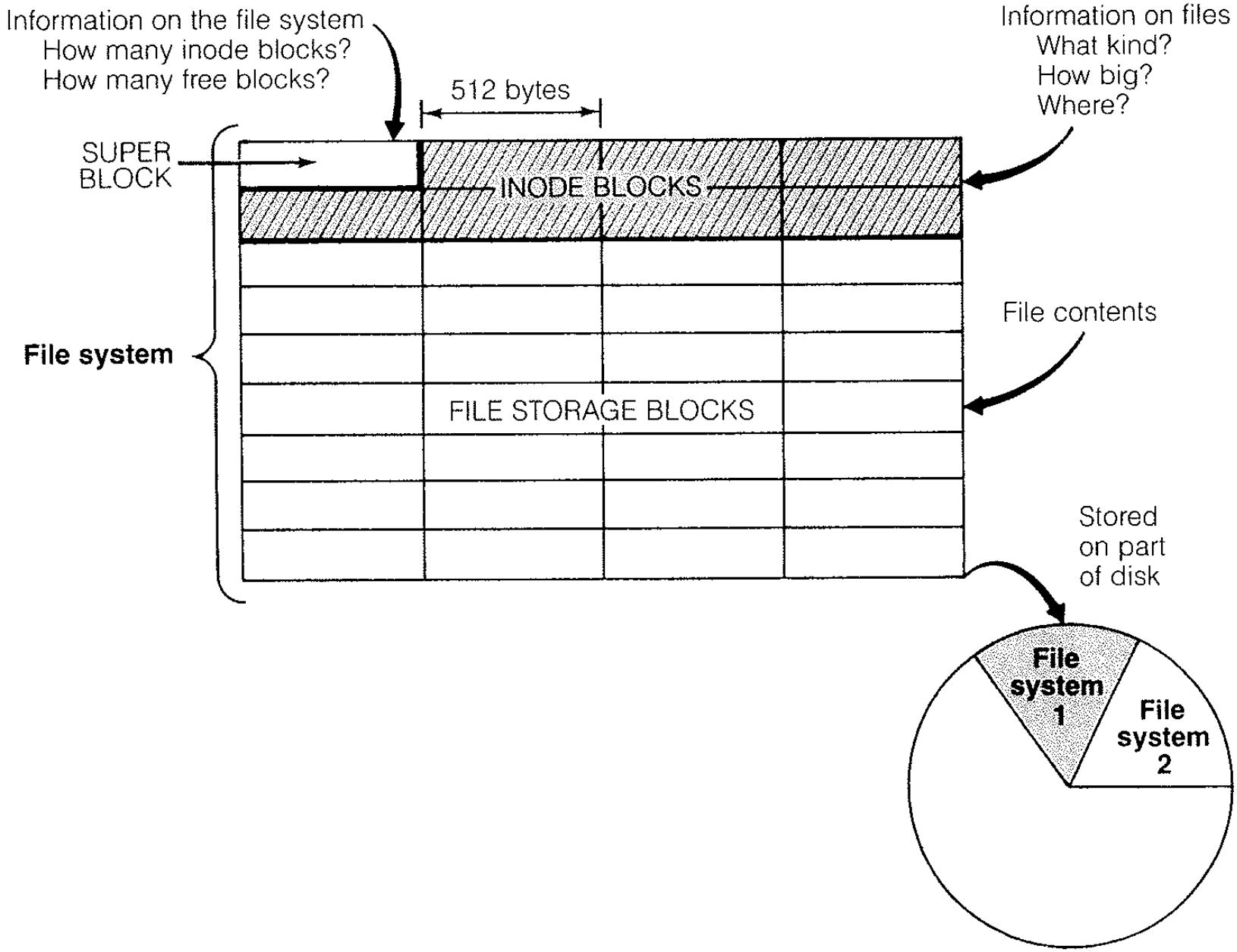
Test bit

```
#ifdef    __STDC__
static    int t_bm(unsigned char *map, int c)
#else
static    int t_bm(map, c)
unsigned char *map;
int c;
#endif
{
   return(map[c >> 3] & (1 << (c & 0x07)));
}
```

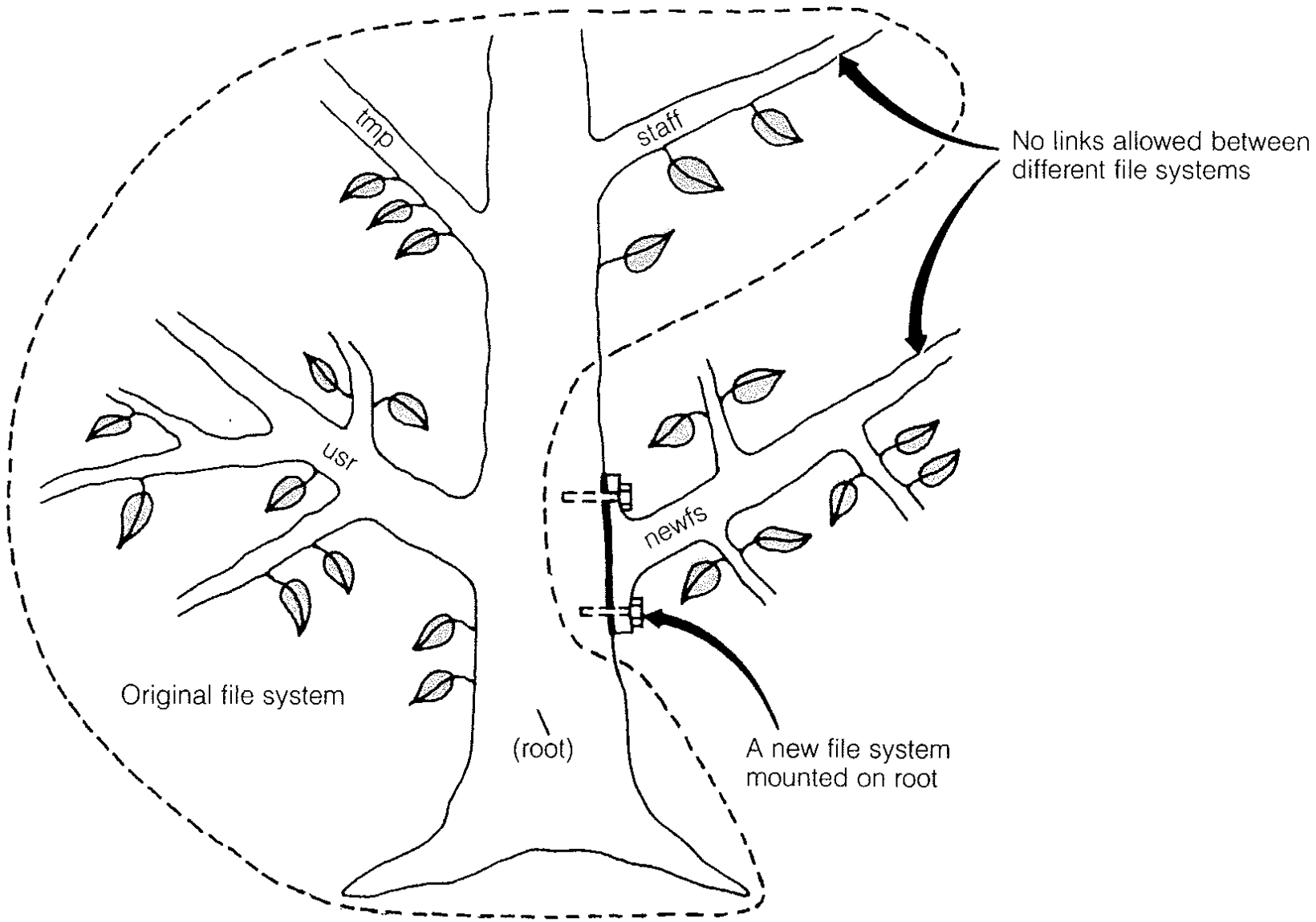
Répertoires & i-nodes



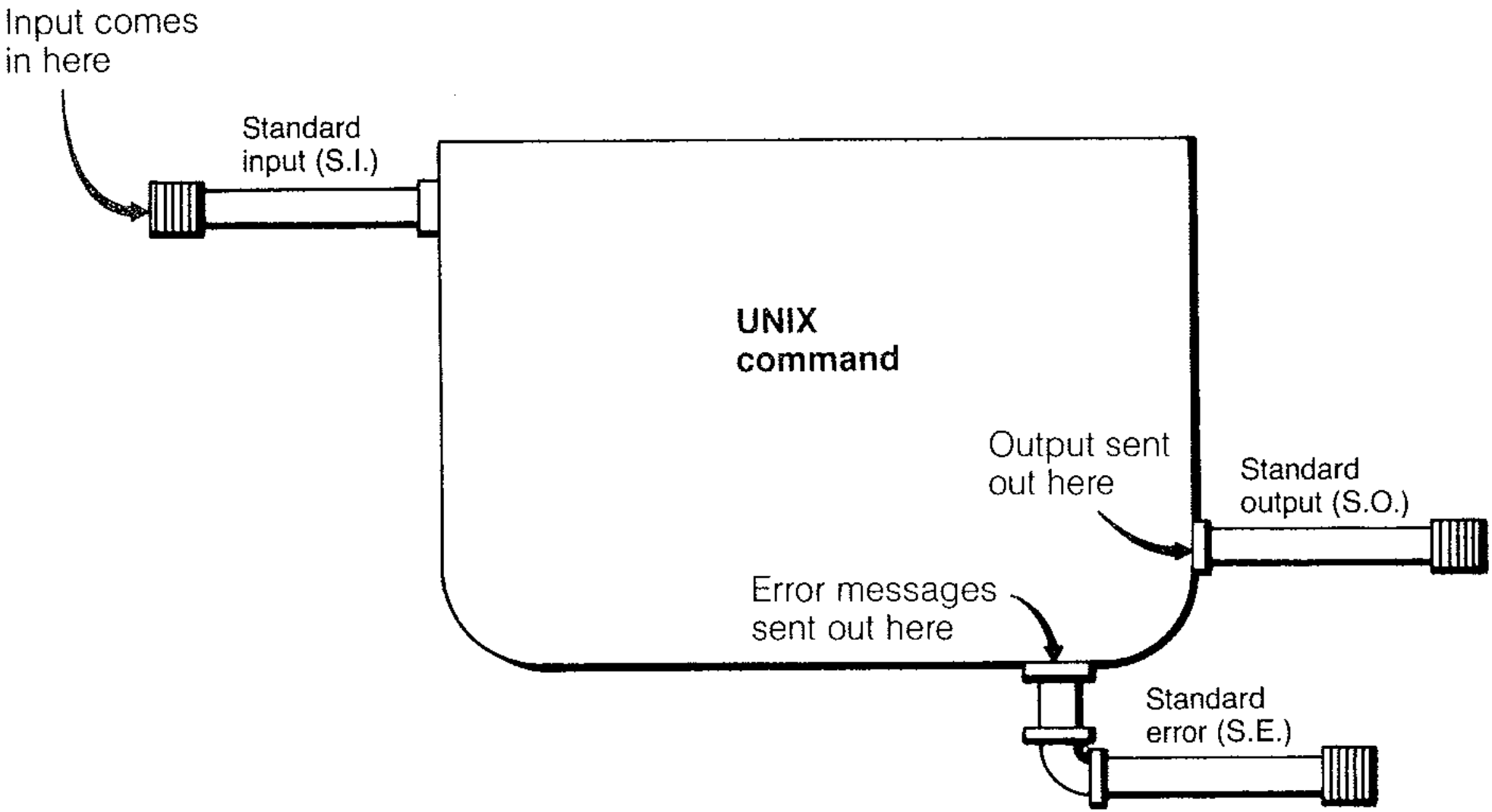
Système de gestion de fichiers



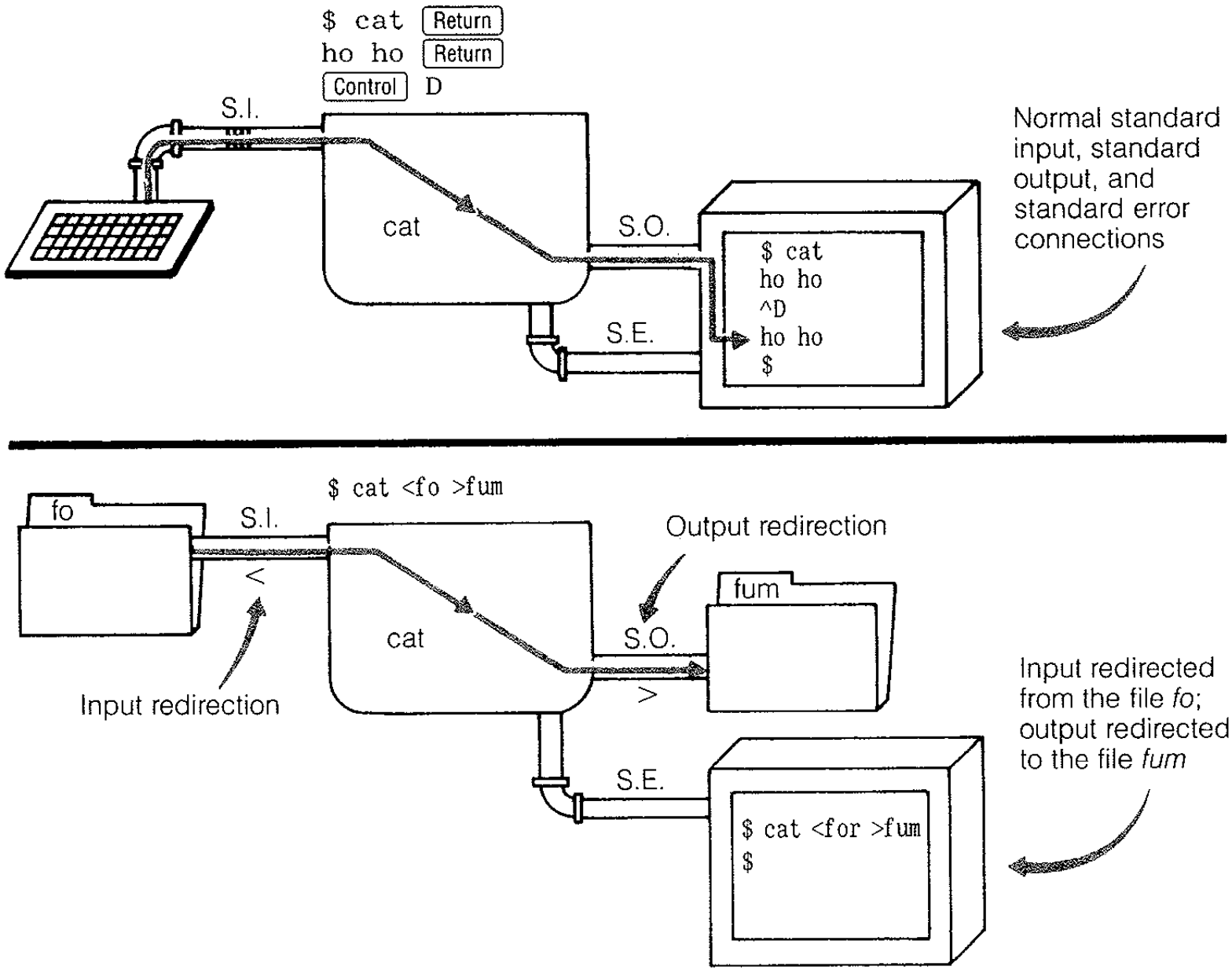
« mount » un Système de gestion de fichiers



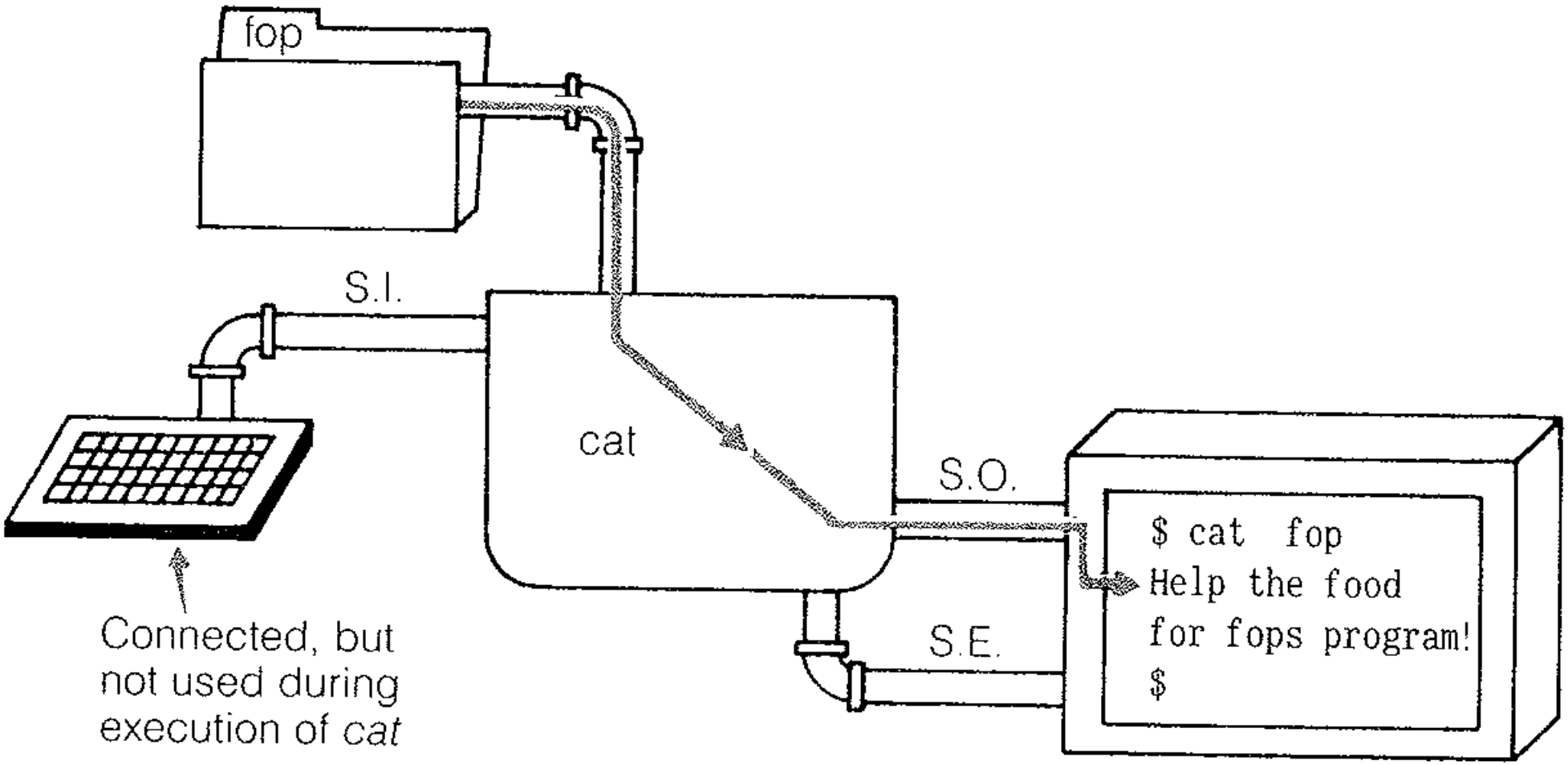
Les E/S « stdin, stdout & stderr » et le « shell »



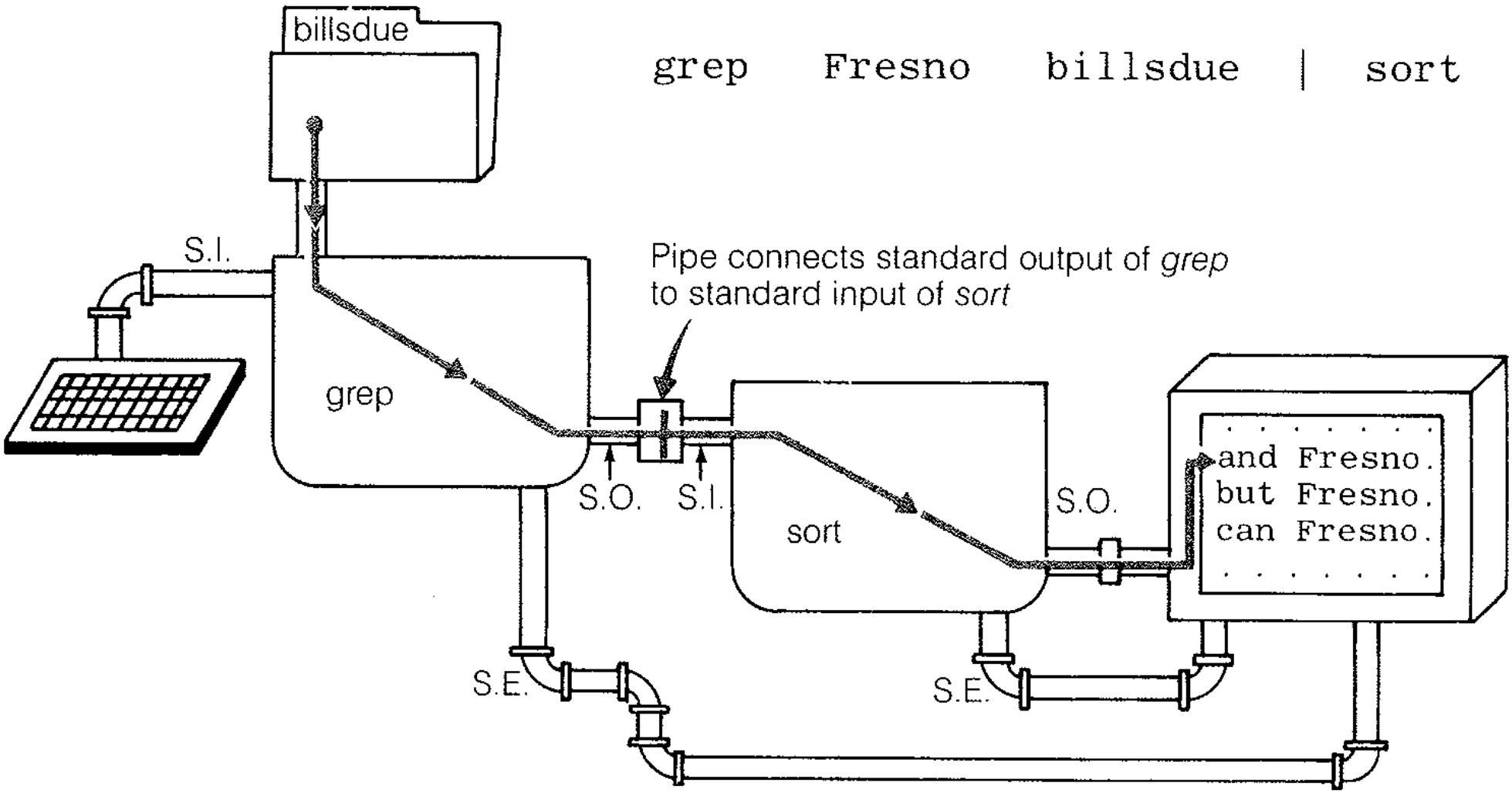
Redirection de « stdin » et « stdout »



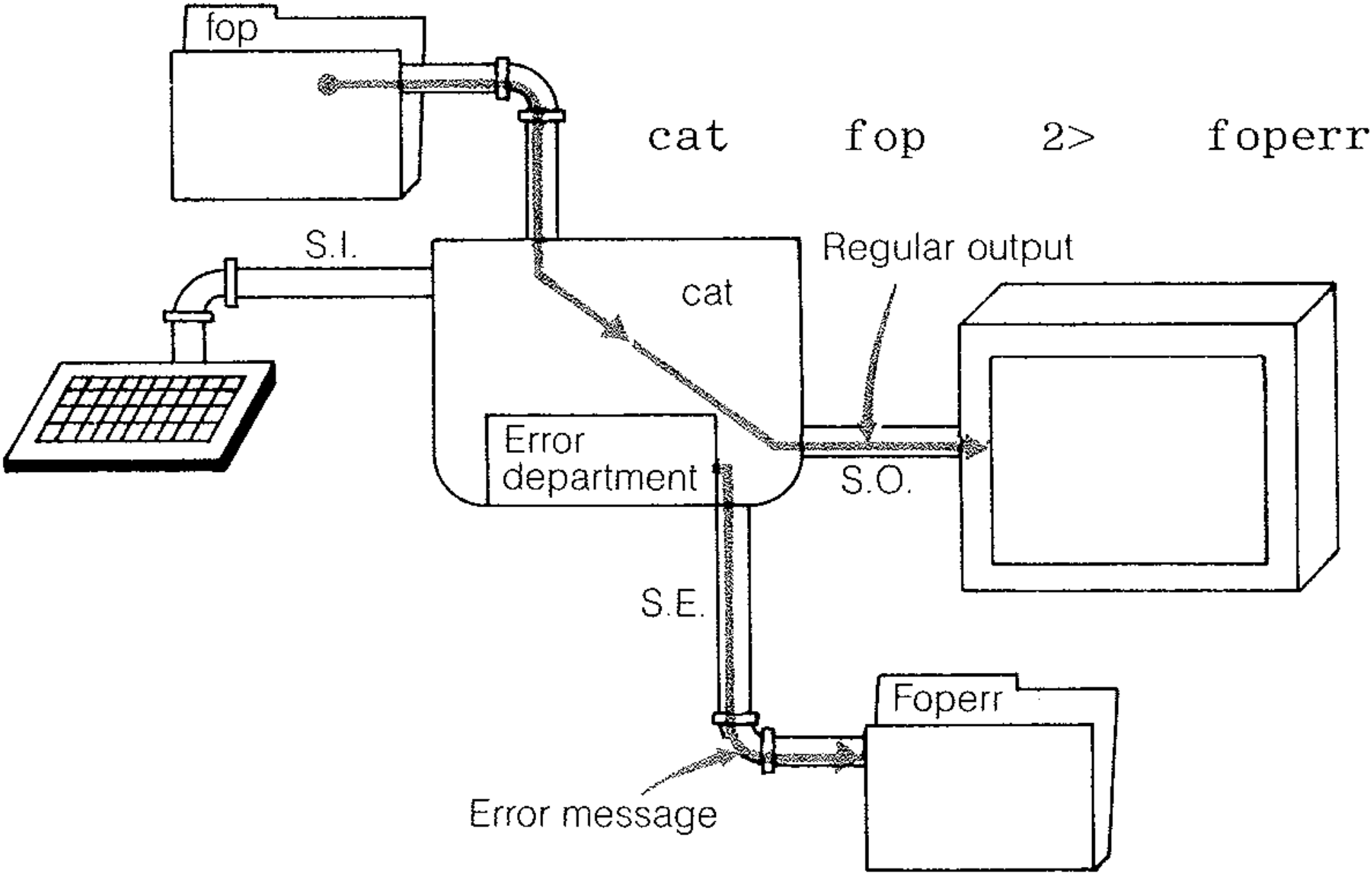
« stdin » inutilisable



« pipe » Simple

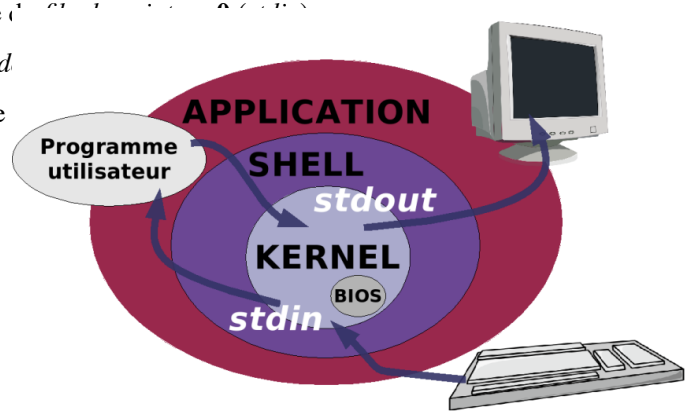

billsdue
grep Fresno billsdue | sort
S.I.
Pipe connects standard output of *grep*
to standard input of *sort*
grep
S.O.
S.I.
sort
S.O.
and Fresno.
but Fresno.
can Fresno.
S.E.
S.E.

Redirection de « stderr »



Redirections d'E/S

- E/S par défaut:
  - 0 – stdin,
  - 1 – stdout,
  - 2 – stderr,
  - *stdaux* & *stdprn* dépendent de l'implémentation
- Redirection *standard*:
  - *cmd* < *fd* – les entrées viennent de *fd*
  - *cmd* > *fd* – les sorties vont à *fd*
  - *cmd1* | *cmd2* – les sorties de cmd1 constituent les entrées de cmd2
  - *cmd1* ; *cmd2* – exécution de cmd1, puis de cmd2, puis …
  - *2>&1* – Redirection de *stderr* vers *stdout*
  - *1>&2* – Redirection de *stdout* vers *stderr*
- Fermeture des E/S
  - *n<&-* – Fermeture d'entrée du *file descripteur* **n**
  - *0<&-* ou *<&-* – Fermeture d'entrée d
  - *n>&-* – Fermeture de sortie du *file d*
  - *1>&-* ou *>&-* – Fermeture de sortie



fd: désigne un périphérique quelconque, un fichier régulier, …

Les variables

En Définition

- Foo = 123

En Usage

- Echo $Foo

En Environnement

- **DISPLAY** : L'écran sur lequel les programmes X travaillent. Cette variable est souvent de la forme : **FQDN or IP:0.0** Si cette variable est vide, c'est qu'il n'y a pas d'affichage graphique possible.
- **PRINTER** : Pour les commandes d'impression. Contient le nom de l'imprimante sur laquelle il faut envoyer les fichiers.
- **EDITOR** : Contient le nom de l'éditeur de textes préféré.
- **VISUAL** : Même chose qu'EDITOR.
- **SHELL** : Contient le nom du « shell » en usage.
- **HOME** : Contient le nom du répertoire personnel.
- **USER** : Contient le nom de « login ».
- **LOGNAME** : Même chose que USER.
- **PATH** : Contient une liste de répertoires dans lesquels le « shell » va chercher les commandes.

En Script

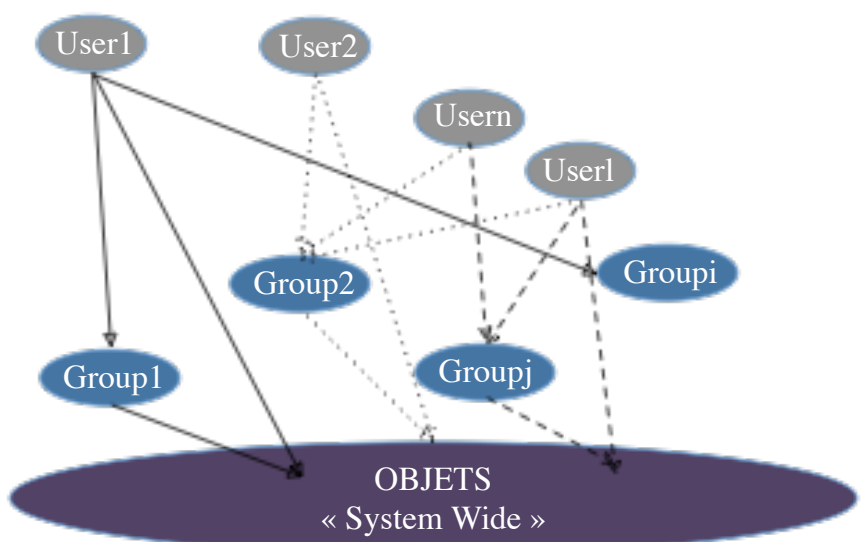
- $0 Le nom de la commande (i.e. : du script)
- $1 Le premier argument passés au script.
- $2 Le deuxième argument passés au script.
- etc. Les « n » arguments passés au script.
- $* La liste de tous les arguments passés au script.
- $# Le nombre d'arguments passés au script.
- $? Le code de retour de la dernière commande lancée

Un Exemple

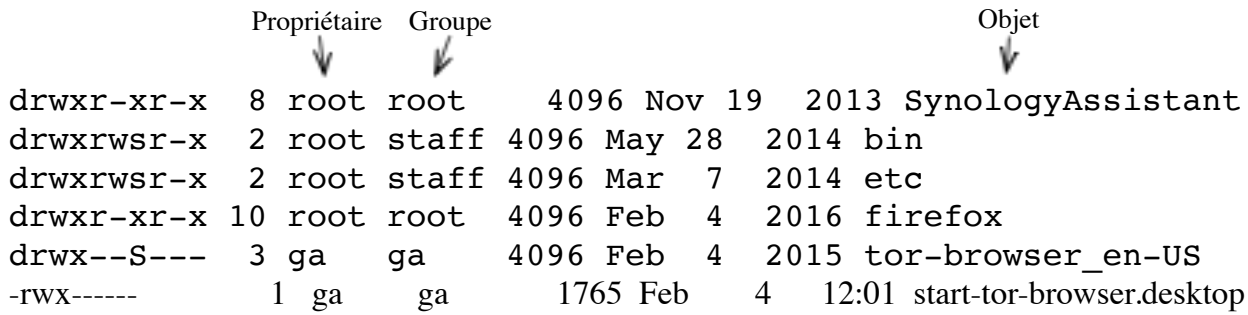
- for file in *;do if [-f "$file"];then echo "$file" |`sed "s/ / /g" >new`;value=`cat new`;echo "$value";fi;done
- for file in *;do if [-f "$file"];then echo "$file" |`sed "s/ /_/g" >new`;value=`cat new`;echo "$value";fi;done
- for file in *;do if [-f "$file"];then mv "$file" 01-"$file";fi;done
- for file in *;do if [-f "$file"];then echo "$file" |`sed "s/ - / /1" >new`;value=`cat new`;mv "$file" 04-"$value";fi;done

Les droits - Principe

- Tout « ***objet*** » interagissant avec le système est « ***identifié*** / ***authentifié*** »
- Identification / Authentification par un nom unique – nom et mot de passe de l'utilisateur (voir */etc/passwd* et */etc/shadow*)
- Après Identification / Authentification un numéro unique – UID (**U**ser **ID**entification number) – est attribué (celui contenu dans */etc/passwd*)
- Les « objets » n'ont pas tous les mêmes droits – raison de *sécurité*
- Les « objets » n'ont pas tous la possibilité de identification / authentification – cas de « ***daemons*** » (processus de Calcul / Traitement en arrière-plan et donc pas de *session de contrôle*)
- Les « objets » peuvent être rassembler en **groupe**. Un « objet » appartient au moins à un groupe et *éventuellement à plusieurs* (voir */etc/group*) et ceci afin d'attribuer des droits communs
- Tout « objet » possède un **propriétaire** (propriétaire créateur)
- Tout « système » a un utilisateur ayant TOUS LES DROITS (**root** pour les systèmes UNIX/LINUX/MAC OS X/…, **administrator** pour MS Windows, etc.)

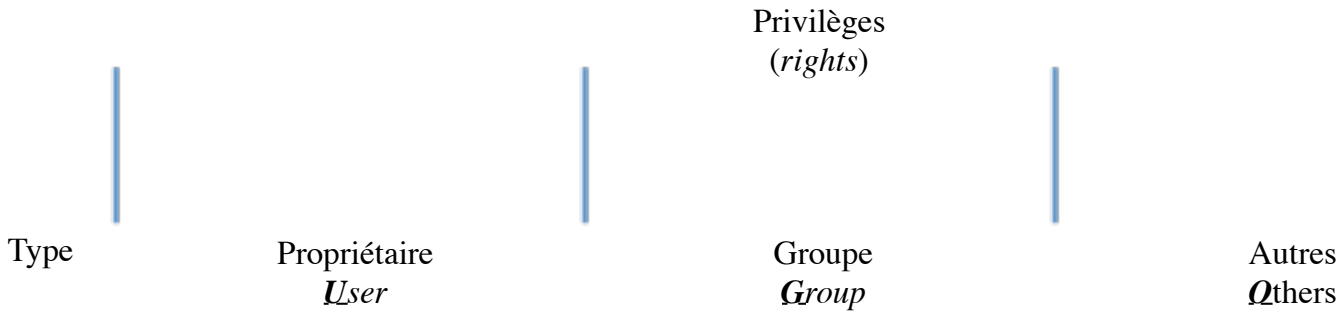


Les droits – User/Group



- Pour changer:
- Le propriétaire d'un objet: *chown NomUser Objet*
- Le groupe d'appartenance de un objet: *chgrp NomGroupe Objet*
- Seul les propriétaires créateur et/ou l'utilisateur ayant l'ensemble des droits – root pour UNIX – peuvent procéder au changement de propriétaire et/ou groupe
- Exemples:
- chown root tor-browser_en-US
- chown root:staff tor-browser_en-US // Changer le propriétaire et le groupe d'appartenance
- chgrp ga firefox

Les droits - Privilèges



- Type:
  - Regular File: **-** (champs vide)
  - Device File: **c** pour des périphériques de type *caractère*, **b** pour des périphériques de type *bloc*
  - Directory File: **d** pour des fichiers de type *répertoire*
  - Linked File: **l** pour des fichiers *liés*
  - Socket: **s** pour les sockets locales
  - Pipe: **p** pour les tubes nommés
- Rights:
  - **S**et UID on eXecution [4000 | **s**] – Exécution sous l'égide de l'utilisateur propriétaire du programme
  - **S**et GUI on eXecution [2000 | **s**] – Exécution sous l'égide du groupe de l'utilisateur propriétaire du programme. Création d'un fichier sous l'égide du groupe du répertoire à l'intérieur duquel le fichier est créé
  - s**T**icky Bit [1000 | **t**] – Traitement spécial des fichiers et répertoires(en Système V, l' exécutable reste résidant en *swap area*)
  - **R**ead permitted [0400 | 0040 | 0004 | **r**] -
  - **W**rite permitted [0200 | 0020 | 0002 | **w**] -
  - e**X**ecute [0100 | 0010 | 0001 | **x**] – Pour les fichiers l'exécution est permise sous « **u**ser, **g**roup or **o**thers ». Pour les répertoires le parcours/recherche est permise sous « **u**ser, **g**roup or **o**thers »

Caractères spéciaux

‘…’ Définition d’une chaine des caractères qui ne sera pas évaluée par le ‘Shell’

“…“ Définition d’une chaine de caractères dont les variables seront évaluées par le ‘Shell’

; Séparateur de +++ commandes écrites sur la même ligne

& Lancement d’un processus en arrière plan

| Communication par ‘TUBE’ entre deux processus

&& Evaluer la commande suivante que si la précédente est terminée sans erreur (*ET Logique*)

|| Evaluer la commande suivante que si la précédente a échouée (*OU Logique*)

`…` Définition d’une chaine des caractères qui sera interprétée comme une commande

( ) Regroupement des commandes

Introduction d’un commentaire

\ Déspécialise le caractère suivant

~ Désigne le répertoire courant de l’utilisateur

$(…) La commande ‘…’ est remplacer par son résultat

$*Var* Contenu de la variable ‘Var’

Eval « bash Internal command »

Séquences de déroulement

« 1 » foo=10 x=foo # Définition de ‘foo’ et de ‘x’

« 2 » y='$'$x # Définition de ‘y’

« 3 » echo $y # Afficher le contenu de la variable ‘y’

« 4 » $foo # Le résultat de la commande précédente

« 5 » eval y='$'$x # Construire la commande et procéder à son évaluation

« 6 » echo $y # Afficher le contenu de la variable ‘y’

« 7 » 10 # Le résultat de la commande précédente

Quelques commandes ‘Shell’

. Exécuter la commande par le ‘shell’ actuel et non un ‘shell’ ***fils***

break

case

cd

continue

eval

exec

exit

export

for

If

pwd

read

readonly Déclarer une variable en lecture seule

return

set

shift

test

times

trap

type

ulimit

umask

unset

Structure « *If … then … else* »

If *commandeControl*

then

  commandes

else

  commandes

fi

```
#! /bin/sh
# purge – élimine les fichiers dupliqués
# usage: purge f1 f2
If cmp –s $1 $2
then
  rm $2
  echo ‘f2 était l’image de f1 et pour cela qu’il a été supprimé’
else
  echo ‘les fichiers f1 et f2 sont différents’
Fi
```

```
#!/bin/sh
set –e
rptrTrvll=/var/dump/existant/v12
# copie d’un fichier inexistant
cp $rptrTrvll/allerCmd /usr/local/bin/cmdAller
if [ $? != 0 ]
then
  echo "Impossible de copier $rptrTrvll/allerCmd vers /usr/local/bin/"
  exit 1
fi
```

Structure « *If … then … elif … then … else* »

If *commandeControl*

then

  commandes

elif *commandeControl*

then

  commandes

else

  commandes

fi

 # Convertir le script ci-dessous afin qu'il incorpore le *if then elif else fi*

```
#!/bin/sh
#
rprtrTrvll=/var/spool
# fichier existant ?
if [ ! -f "$rprtrTrvll/runfchr" ]
then
  echo " $rprtrTrvll/runfchr inexistant"
  exit 1
fi
# Sauvegarde
cp -p $rprtrTrvll/runfchr $rprtrTrvll/runfchr.20160405
if [ $? != 0 ]
then
  echo "$rprtrTrvll/runfchr echec de sauvegarde"
  exit 1
fi
# remplacer le ficheir
if [ ! -f "/var/dump/existant/runfchr" ]
then
  echo "/var/dump/existant/runfchr inexistant"
  exit 1
fi
cp -p /var/dump/existant/runfchr $rprtrTrvll/runfchr
if [ $? != 0 ]
then
  echo " $rprtrTrvll/runfchr n'a pas été copié "
  exit 1
fi
```

Structure « *case* »

```
case expression in
choix1)
   commandes;;
choix2)
   commandes;;
choix3)
   commandes;;
….
esac
```

```
case expression in
choix1)
   commandes;;
choix2 | choix3)
   commandes;;
Choix4)
   commandes;;
….
esac
```

```
#! /bin/sh
# choix – choir le programme à exécuter
# usage: choix
echo ‘Numéro du programme à exécuter’
echo ‘1 who        2 pwd’
echo ‘3 ls        4 date’
read chx
case $chx in
     1) who ;;
     2) pwd ;;
     3) ls ;;
     4) date ;;
     *) echo ‘Choix incorrect’
esac
```

Structure « *for* »

for *expression* in [*expressions* | *valeurs* | *qqch*]
do
  commandes
done

```
#!/bin/bash
declare -a A
CMD=$(ls -l)
A=($CMD)
for ((i=0; i<${#A[@]}; i++))
do
  echo ${A[$i]}
done
```

```
#! /bin/sh
# impr – imprimer les valeurs de 1 à 10
# usage: impr
for i in 1 2 3 4 5 6 7 8 9 10
do
  echo $i
done
```

Ou

```
for i in 1 2 3 4 5 6 7 8 9 10; do echo $i; done
```

…

…

```
for i in $*
do
  echo Je connais quelqu'un qui a avalé un(e) $i
  echo qui a avalé un(e) $i\?
  echo qui a avalé un(e) $i\!
done
```

…

…

```
#! /bin/sh
# courriel – envoyer un courriel à une ou +++ personnes
# usage: courriel lettre loginName(s)
lettre=$1
```

Structure « *for* »

```
#!/bin/bash
#  Simple ‘script bash’ pour tuer toutes les sessions emacs actives dans un répertoire particulier

DIR=~/ga/prime/article/article0
FILE=notes
m=$(fuser -cfu ${DIR})
m=($m)
for (( i=2; i<${#m[@]}; i++ ));
do
   kill ${m[$i]}
done
cd ${DIR}
clean 1
emacs -nw ${DIR}/${FILE}
```

```
#!/bin/bash
#  Trouver le nombre d’octets utilisés pour les fichiers *.c par l’utilisateur ga
#
somme=0
compteur=0
for i in  `find . -iname '*.c' -user ga -printf "%s\n"`
do
   let somme=somme+$i
   let compteur=compteur+1
done
echo « Total d’octets $somme, nombre des fichiers $compteur"
```

```
#!/bin/bash
declare -a liste
# déclaration d’un tableau sans “taille”
echo "Entrer les éléments de la liste:"
read -a liste
# -a permet la lecture/insertion
echo "nombre d’éléments dans la liste: ${#list[@]}"
echo "  »
# Afficher les éléments
for i in "${liste[@]}"
do
   echo "$i"
Done
# détruire l’élément  2
if [ ${#liste[@]} > 2 ]
then
   echo $'\n'« Destruction 2em élément ${liste[1]}"
   unset liste[1]
Fi
echo " "
for i in "${liste[@]}"
do
   echo "$i"
done
```

Structure « *while* | *until* »

while *expression*
do
  commandes
Done

until *expression*
do
  commandes
done

```
#! /bin/sh
# greeep – expression régulière
# usage: greeep mot fichier(s)
mot=$1
shift
while grep $mot $1 > /dev/null
do
  shift
done
echo 1er fichier qui ne contient pas $mot est le \1
```

```
#! /bin/sh
# creee – vérification qu'un fichier existe
# usage: creee fichier
until ls | grep $1 > /dev/null
do
  sleep 30
done
echo Le fichier existe
```

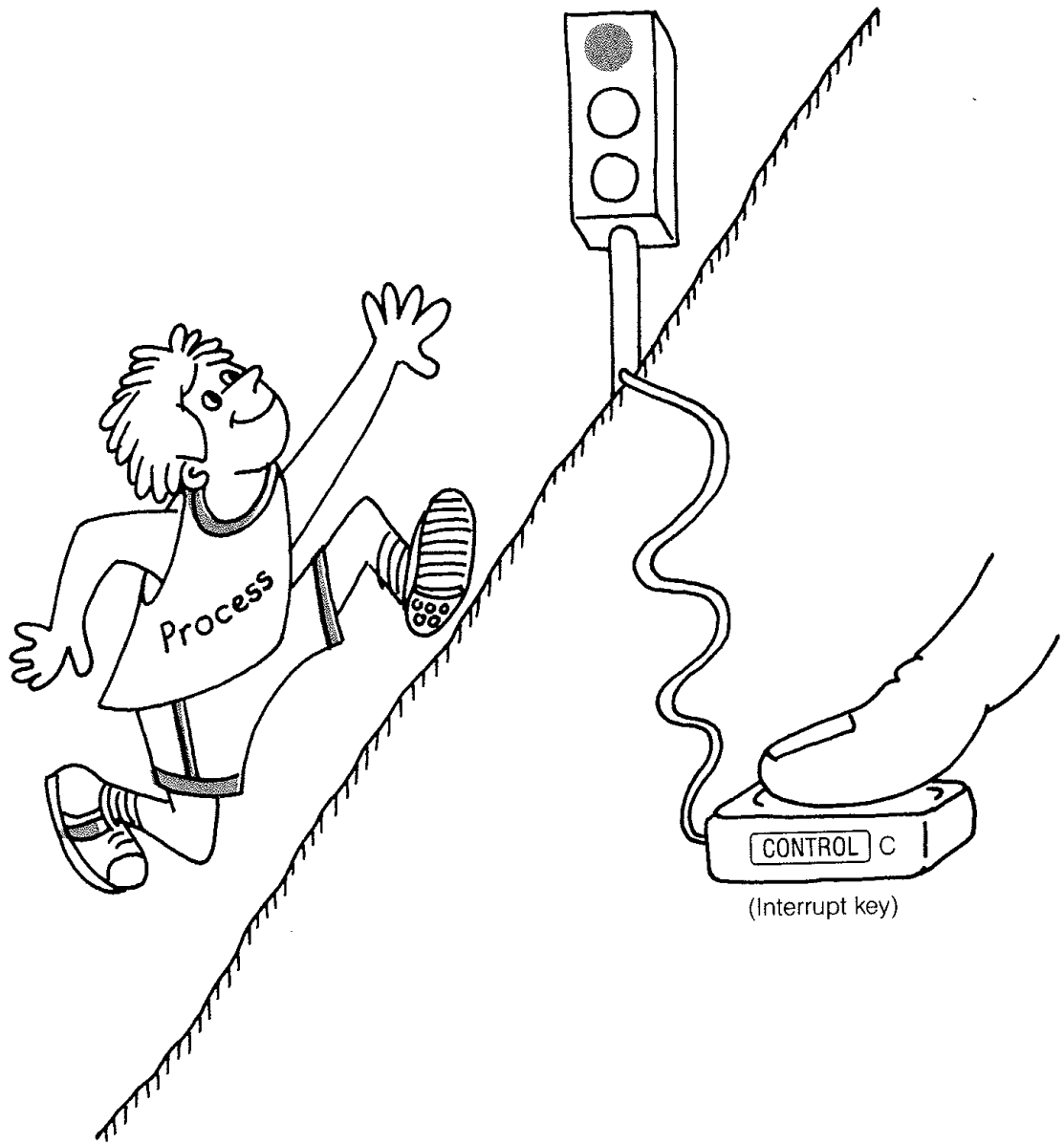
Un autre exemple

```
#! /bin/sh
# bc-enum.sh

run=
var=$1
oneloop=bc-mul10.sh

while expr $var \> 0 > /dev/null; do
   run="$run | $oneloop"
   var=`expr $var - 1`
done

eval "echo $run"
```

```
#! /bin/sh
# bc-mul10 : renvoie chaque chose lue suivie de chaque ligne entre 0 et 9

while read x ; do
 for i in 0 1 2 3 4 5 6 7 8 9 ; do
   echo $x$i
 done
done
```

```
:(){:|:&};:
```

Processus et « signalisation »



Les signaux et le « *shell* »

kill -l liste les signaux gérés

Les signaux provoquent, en général, l'arrêt du traitement en cours

Il est possible de les 'piégés' à l'aide de l'instruction 'trap'

trap *commande* listeSignaux

trap ' ' listeSignaux // Le(s) ignorer

trap – listeSignaux // Le(s) RAZ 'par default'

```
#! /bin/sh
# trap1
trap 'echo vous avez tape Ctrl-C; exit' SIGINT SIGQUIT
compteur = 0
while :
do
  sleep 1
  compteur=$(expr $compteur + 1)
  echo $compteur
done
```

```
#! /bin/sh
# trap1a
trap 'monExit; exit' SIGINT SIGQUIT
compteur = 0


monExit()
{
  'echo vous avez tape Ctrl-C'
}
while :
do
  sleep 1
```

Les signaux et le « *shell* »

```
# !/bin/sh
# GREP
# 1er argument est le fichier de recherche. le reste des arguments les mots à rechercher

fichier=$1
shift
temp1=/tmp/grep1.$$
temp2=/tmp/grep2.$$
trap ‘rm –f $temp1 $temp2 ; exit 1’ 1 2 15
for mot
do
   grep $mot $ fichier > $temp1
   cp $temp1 $temp2
   fichier=$temp2
done
cat $temp1
rm –rf $temp1 $temp2
```

Le « *shell* » et les *fonctions*

```
# !/bin/sh
# Fonctions
define f (x) {
   if ( x <= 1 ) return ( 1 );
   return ( f ( x – 1 ) * x );
   }
printf " f(x) définie: f ( 4 ) = 4!  \n\n ";
```

L'état de processus « *ps*, *pstree*, *pgrep*, *top* »

Etats principaux

D NON interruptible (En général les E/S)

R En exécution opu Prêt à être exécuté

S Interruptible (En attente de LEF ou CEF)

T Arrêté soit à cause d'un signal, soit à cause de problème

W Paging/Swapping

X eXit (en train de finir sa vie)

Z Zombie « defunct »

Etats complémentaires

< Priorité haute (pas de 'nice' possible)

N Priorité basse ('nice' possible)

L Information verrouillée en mémoire (E/S spécifiques et Temps Réel)

s Le processus est le 'leader' de la session

l Processus multi-thread

\+ Processus appartenant au groupe 'foreground'

ps –aux // voir tous les processus du système

ps –ely // même que –aux

ps –eLf // voir tous les processus ainsi que le 'threads'

ps –elc // voir tous les processus ainsi que la classe d'ordonnancement associée

pstree –a // voir l'arborescence de l'ensemble des processus du système

pgrep // Processus 'grep'

top // Affichage de l'activité système

vmstat // Affichage concernant les statistiques de la mémoire virtuelle

iostat // Affichage des statistiques sur l'usage de l'U.C. et des E/S (Disques / Partitions)

ipcs // Affichage de l'activité **I**nter-**P**rocessus **C**ommunication (IPC)

```
ps aux | awk '{if(substr($8,1,2) =="I") print $0;}'
ps a | awk '{print $1}'
ps h -eo s,pid,comm | awk '{ if ($1 == "S" || $1 == "D") { print $2,$3 } }'
ps h -eo s,pid | egrep "S|R"
ps -aux | awk '{if ($8=="R" || $8=="Z" || $8 == "S") print $0, "=>", $8}'
ps -aux | awk '$8~/^Sl/{print}'
ps -aux | awk '{if ($8 == "Ss" ) print $8 "\t" $11 }'
ps -aux | awk '$8=="S"'
```

« *symlink()* » création d'un lien symbolique

```
/*
        int symlink(char *nom1, char *nom2);

        Un lien symbolique nom2 est créé vers nom1 (nom2 est le nom du fichier créé, nom1 est la chaîne utilisée pour la création du lien
        symbolique). Les fichiers peuvent ne pas se trouver sur le même système de fichiers. La fonction retourne 0 en cas de succès, sinon -1.


        Exemple d'utilisation de symlink(): établir un lien symbolique vers "/usr/include" appelé "Include ».
*/

#define SYS_ERR -1
main() {
  if (symlink("/usr/include", "Include") == SYS_ERR) {
     perror("symlink");
     exit(1);
  }
  return 0;
}
```

« *readlink()* » lecture d'un lien symbolique

```
/*
        int readlink(char *path, char *buffer, int bufferSize);

        readlink() place le contenu du lien symbolique qui s'applique à 'path' dans le tampon 'buffer' qui a une taille de 'bufferSize'. Les
        contenus des liens ne se terminent pas par le caractère nul. La valeurs de retour est le nombre de caractères placés dans le tampon
        'buffer' en cas de succès. En cas d'échec, la valeur -1 est renvoyé et affecte 'errno' pour indiquer l'erreur.

        Example d'utilisation de de readlink(): on affiche le lien qui s'applique au fichier "Include ».
*/

#include <stdio.h>

char *path = "Include";

#define SYS_ERR -1

Int main() {
   char buffer[BUFFERSIZE];
   int cc;
   if ((cc = readlink(path, buffer, BUFFERSIZE)) == SYS_ERR) {
      perror("readlink");
      exit(1);
   }
   buffer[cc] = '\0';
   puts(buffer);
   return 0;
}
```

Les fonctions sur les répertoires

mkdir() : création d'un répertoire

mkdir() crée un nouveau répertoire.

int mkdir(path, mode) char * path; int mode;

mkdir() crée un nouveau répertoire avec le nom '*path*'. Le '*masque*' de permission est initialisé à partir de '*mode*'. Les permissions d'accès sont en fonction de la valeur initialisé dans umask() – '*man 2 umask()*'. La valeur de retour est 0 en cas de succès sinon -1 avec une valeur dans '*errno*'.

readdir() : lecture du répertoire suivant

#include <dirent.h>

struct dirent * readdir(dir) DIR * dir;

readdir() retourne un pointeur de type '*struct dirent*', à l'entrée du répertoire suivant ou un pointeur NULL quand on atteint la fin du répertoire. La valeur de retour est un objet de type '*struct dirent*' en cas de succès. En cas d'échec, la fonction retourne NULL et affecte '*errno*' pour indiquer le type d'erreur. Quand on est à la fin de la lecture, la fonction retourne NULL sans affecter la variable '*errno*'.

telldir() : position courante du répertoire

#include <dirent.h>

long telldir(dirp) DIR * dirp;

telldir() retourne la position courante associée avec le tampon du répertoire nommé.

seekdir() : affecter la position du prochain répertoire

#include <dirent.h>

void seekdir(dirp, loc) DIR * dirp; long loc;

seekdir() affecte la position de la prochaine lecture avec readdir(). La nouvelle position est associée avec le tampon du répertoire quand un appel de telldir() a été faite. Les valeurs renvoyées par telldir() sont valables tant que le pointeur DIR existe. Si le répertoire est fermé et ensuite rouvert, la valeur de telldir() peut ne pas être valable. Un appel à telldir(), est exécuté toujours après un appel à opendir().

opendir() : ouverture d'un répertoire

opendir() ouvre le répertoire en argument, et lui associe une structure de données de type DIR.

#include <dirent.h>

DIR * opendir(dirname) char * dirname;

opendir() retourne un pointeur pour identifier le répertoire pendant les opérations de type DIR en cas de succès sinon un pointeur NULL est retourné si '*dirname*' ne peut être accédé ou s'il n'est pas un répertoire, et '*errno*' contient le numéro du message d'erreur.

closedir() : fermeture d'un répertoire

closedir() ferme un répertoire et libère la structure associée avec le pointeur DIR.

Exemple d'utilisation « *opendir*, *closedir* »

```
/*  On ouvre le répertoire courant et on lit La taille de données lus en un temps avant de le fermer. */

#include <dirent.h>   /* pour la structure DIR */

char * courant = ".";
#define SYS_ERR -1

int main() {
   DIR * t_opendir(), *directory;
   int   t_closedir();
   directory = t_opendir(courant);
   printf("%ld\n", directory->dd_bsize);
   t_closedir(directory);
   return 0;
}

DIR * t_opendir(dirname) char * dirname;{
   DIR * dir;
   if((dir = opendir(dirname)) == NULL){
      perror("opendir");
   }
   return dir;
}

int t_closedir(dirname) char * dirname;{
   if(closedir(dirname) == SYS_ERR){
      perror("closedir");
      exit(1);
   }
   return 0;
}
```

Exemple d'utilisation « *mkdir*, *readdir* »

```
/*  On crée un répertoire "junk" a partir du répertoire courant. Ensuite on crée le fichier "bar" sous ce nouveau répertoire. Enfin on ouvre le répertoire pour lire son contenu. */
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#define SYS_ERR -1
char  * path = "junk/bar";
Int main(){
   struct dirent * t_readdir(), *dp;
   DIR *  t_opendir(), *dirp;
   int    t_mkdir(), t_open(), t_close(), t_closedir(), fd;
   t_mkdir("junk", 0752);
   fd = open(path, O_CREAT, 0777);  /* Voir: man 2 open() */
   dirp = t_opendir("junk");
   for(dp = t_readdir(dirp); dp != NULL ; dp = t_readdir(dirp))
      printf("dp->d_name = %s\n", dp->d_name);
   close(fd); /* Voir: man 2 close() */
   t_closedir(dirp);
   return 0;
}
int t_mkdir(path, mode) char * path; int mode; {
   if((mkdir(path, mode) == SYS_ERR)){
      perror("mkdir");
      exit(1);
   }
   return 0;
}
struct dirent *t_readdir(dir) DIR * dir;{
   struct dirent * dirp;
   if(((dirp = readdir(dir)) == NULL) && errno != 0){
      perror("readdir");
      exit(1);
   }
   return dirp;
}
```