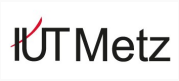
R3.05 - Programmation Système

Fichiers

Cyril Grelier

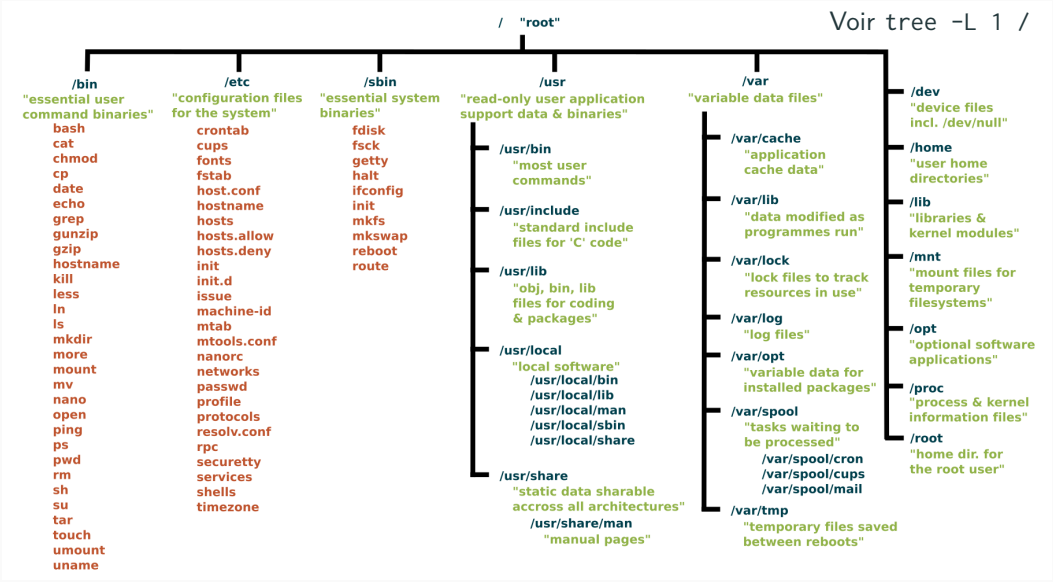
Université de Lorraine - IUT de Metz



Fichiers

- Un **fichier** = **séquence d'octets** sur le disque
- Le système ne connaît pas le sens (texte, image, binaire, ...)
- Les standards de format aident à aident à interpréter le contenu (`.png`, `.mp4`, ...)
- Détection de type : `file mon_fichier`.
- Sous Unix/Linux, **tout est fichier** :
  - fichiers "classiques" (texte, binaire, vidéo, ...)
  - répertoires (/home, /tmp, ...)
  - périphériques (clavier, disque, carte son...)
  - tubes (pipes)
  - sockets
  - liens symboliques
  - ...
- Tous se manipulent via les mêmes appels système : open, `read`, `write`, `close`, ...

Arborescence - Racine



File System

Le **File System** (**Système de fichiers**) est responsable de l'organisation des fichiers sur le disque :

- les inodes, répertoires et blocs disque
- les permissions et métadonnées
- l'accès efficace et sécurisé aux données
- la cohérence des données (journalisation, verrouillage, ...)

Interface entre le matériel (disque physique) et le noyau.

Exemples de FS

- ext4 (Linux)
- NTFS (Windows)
- FAT32 / exFAT
- APFS (macOS)

Chaîne d'accès :

Disque ← Système de fichiers ← inodes/&répertoires ← Noyau ← FD ← Programme

FD : *File Descriptor* - Descripteur de fichier (type : `int`)

Inodes & taille logique vs physique

Inode (*index node* - inœud en français) contient les métadonnées :

- type (fichier, répertoire, lien, ...),
- taille (en octets),
- droits d'accès (lecture/écriture/exécution),
- propriétaire (UID) et groupe (GID),
- dates (création, accès, ...),
- nombre de liens,
- pointeurs vers les blocs disque.

```
1  $ echo "coucou" > mon_fichier.txt
2  $ stat mon_fichier.txt # stat affiche les métadonnées
3    Fichier : mon_fichier.txt
4     Taille : 11               Blocs : 8          Blocs d'E/S : 4096   fichier
5  Périphérique : fc01h/64513d        Inœud : 17406240    Liens : 1
6  Accès : (0644/-rw-r--r--)  UID : (675349/cgrelier)   GID : (200367/optimist)
```

Taille logique (octets utiles, 11 o) $\neq$ **taille physique** (8 blocs de 512 octets soit 4096 o/4 ko)

Types de fichiers & droits

Types (ls -l) :

- `-` : fichier régulier
- `d` : répertoire (directory)
- `l` : lien symbolique (symlink)
- `c` : périphérique caractère (terminal, clavier...)
- `b` : périphérique bloc (disque dur...)
- `s` : socket
- `p` : pipe nommé (FIFO)

```
1  $ ls -l /dev/null /bin/ls /
2  -rwxr-xr-x 1 root root 138216 ... /bin/ls
3  crw-rw-rw- 1 root root 1,3    ... /dev/null
4  drwxr-xr-x 169 root root      ... /etc
```

Droits : `rwxr-xr-x` = proprio `rwx`, groupe `r-x`, autres `r-x`. `chmod`, `chown` pour modifier droits/proprio.

Accès aux fichiers en C

Deux voies :

- **POSIX (unistd/fcntl)** : bas niveau, **non bufferisé**, FD entiers.
  1. Programme (open, close, read, write, ...)
  2. noyau Linux (gestion FD, inodes, FS)
  3. disque
- **Libc (stdio)** : haut niveau, **bufferisé** en espace utilisateur (`FILE*`).
  1. Programme (fopen, fclose, fprintf, fgets, ...)
  2. libc (FILE*, tampon mémoire utilisateur)
  3. appels système POSIX (open, close, read, write, ...)
  4. noyau Linux (gestion FD, inodes, FS)
  5. disque

Bufferisation : réduire les appels système en regroupant les I/O.

Libc : fopen/fprintf/fclose

```c
// includes : stdio.h, stdlib.h


FILE *f = fopen("data.txt", "w");           // Ouverture du fichier
if (!f) {                                   // Check des erreurs à l'ouverture
    perror("fopen");
    exit(EXIT_FAILURE);
}

fprintf(f, "Hello bufferisé!\n");           // Écriture dans le fichier

fclose(f);                                  // Fermeture du fichier
```

POSIX : open/write/close

```c
// includes : fcntl.h, unistd.h, string.h, errno.h, stdlib.h

int fd = open("data.txt", O_WRONLY|O_CREAT|O_TRUNC, 0644);  // Ouverture du fichier
if (fd == -1) {                                 // Check des erreurs à l'ouverture
    perror("open");
    exit(EXIT_FAILURE);
}

const char *msg = "Hello bas niveau!\n";
ssize_t n = write(fd, msg, strlen(msg));    // Écriture dans le fichier
if (n == -1) {                                  // Check des erreurs à l'écriture
    perror("write");
    close(fd);                                  // Fermeture du fichier
    exit(EXIT_FAILURE);
}

close(fd);                                      // Fermeture du fichier
```

Descripteurs de fichiers & redirections

- À l'exec d'un processus : **0=stdin**, **1=stdout**, **2=stderr**.
- Le noyau associe un **int** (FD) à un fichier ouvert.

```
1  $ ls > out.txt      # stdout vers fichier
2  $ ls 2> err.txt     # stderr
3  $ sort < data.txt   # stdin depuis fichier
4  $ ls | grep ".c"    # pipe stdout -> stdin
```

Répertoires & fichiers spéciaux

Sous Unix/Linux, un répertoire est un fichier spécial :

- table d'association entre :
  - nom de fichier
  - numéro d'inode
- un répertoire = une liste (nom → inode)

Autres Fichiers spéciaux

- `/dev/null` : trou noir
- `/dev/urandom` : aléatoire
- `/proc`, `/sys` : interfaces noyau

Contenu d'un répertoire avec deux fichiers :

```
$ ls -laih
total 8,0K
18117141 drwxr-xr-x  2 cgrelier optimist 4,0K oct.  21 13:52 .      <- le répertoire
17301514 drwxr-xr-x 65 cgrelier optimist 4,0K oct.  21 13:52 ..     <- le parent
18117143 -rw-r--r--  1 cgrelier optimist    0 oct.  21 13:52 mon_fichier1.txt
18117145 -rw-r--r--  1 cgrelier optimist    0 oct.  21 13:52 mon_fichier2.txt
```

Liens physiques vs symboliques

- **Lien physique (hard link)** : un second nom pointant vers le même inode. Le fichier reste valide tant qu'au moins un lien existe.
- **Lien symbolique (soft link / symlink)** : un raccourci vers un autre fichier. Si la cible est supprimée, le lien devient cassé.

```
1 $ ln fichier_original lien     # hard link
2 $ ln -s fichier_original lien  # symlink
```