



Systèmes et réseaux

Chapitre 1 Entrées/sorties, fichiers, répertoires



Pablo Rauzy <pr@up8.edu>
pablo.rauzy.name/teaching/sr

Entrées/sorties, fichiers, répertoires

Système de fichiers

- ▶ Il y a deux façons de voir un système de fichier :
 - une façon “externe”, pour ses utilisateur·ices,
 - une façon “interne”, par ses développeur·euses.

La vision “externe”

- ▶ Vous connaissez la vision “externe” du système de fichiers :
 - une arborescence de répertoires contenant des fichiers, avec un répertoire particulier, /, à la racine.
- ▶ Dans cette vision, chaque fichier ou répertoire est identifié par ses *chemins*, relatifs ou absous.

La vision “interne”

- ▶ En interne, un système de fichier est composé de deux types de structures :
 - les *inodes*,
 - les données de contenu.
- ▶ Une inode contient toutes les métas informations sur le fichier :
 - type de fichier (fichier ordinaire, répertoire, lien, périphérique, fifo, socket, ...);
 - date de création, modification, et accès ;
 - propriétaire et groupe ;
 - mode (droit d'accès en lecture, écriture, exécution, setuid, etc.) ;
 - périphérique de stockage, taille, nombre de blocs, ...
- ▶ Les données de contenu... sont le contenu des fichiers.
 - Pour un répertoire il s'agit d'une liste de noms et de l'inode associée.

Manipulation de fichiers

► Lorsqu'un programme veut manipuler un fichier :

- il demande au système via l'interface "externe" du système de fichier à accéder à un fichier donné ;
- le système garde en mémoire côté noyau que ce fichier (inode mémoire, global au système) est ouvert par ce programme (entrée dans la table des fichiers ouverts, global au système, + entrée dans la table de descripteurs de fichiers, local au programme) ;
- le programme, côté utilisateurice, dispose simplement d'un *descripteur de fichier* : un petit entier identifiant le fichier en question.

► Les manipulations se font ensuite via des appels systèmes.

- Ou bien sûr, via des fonctions de plus haut niveau faisant elles-mêmes usage des appels systèmes.
- Exemple, les fonctions manipulants des FILE *.

► Par défaut au lancement d'un programme, trois descripteurs de fichiers sont ouverts :

- 0 pour son entrée standard,
- 1 pour sa sortie standard,
- 2 pour sa sortie d'erreur standard.

Appels systèmes

- ▶ Il y a de nombreux appels systèmes liés à la manipulation de fichiers.
 - La plupart existent en plusieurs variantes, qui dépendent essentiellement du type de leurs arguments, pour des raisons pratiques.
- ▶ Les appels systèmes sont largement documentés dans les pages de manuel.
 - Pour éviter de simplement paraphraser la documentation complète, nous allons nous y référer directement.
 - Pensez à faire aussi attention aux sections RETURN VALUE, ERRORS, et SEE ALSO.
 - Pensez également à vous référer aux pages de manuels de la section 3type.

Entrées/sorties bas niveau

- ▶ `creat` : créer ou tronquer un fichier
 - `#include <fcntl.h>`
 - `int creat(const char *pathname, mode_t mode);`
- ▶ `open` : ouvrir et possiblement créer un fichier
 - `#include <fcntl.h>`
 - `int open(const char *pathname, int flags, mode_t mode);`
- ▶ `close` : fermer un descripteur de fichier
 - `#include <unistd.h>`
 - `int close(int fd);`
- ▶ `read` : lire depuis un descripteur de fichier
 - `#include <unistd.h>`
 - `ssize_t read(int fd, void buf[.count], size_t count);`
- ▶ `write` : écrire vers un descripteur de fichier
 - `#include <unistd.h>`
 - `ssize_t write(int fd, const void buf[.count], size_t count);`
- ▶ `lseek` : repositionner le curseur d'un fichier
 - `#include <unistd.h>`
 - `off_t lseek(int fd, off_t offset, int whence);`

Entrées/sorties haut niveau

- ▶ Les entrées/sorties de haut niveau en C sont implémentées par la bibliothèque **stdio**.
 - On ne manipule plus directement des descripteurs de fichiers mais des pointeurs vers des structures **FILE** qu'on appelle des *flux* (*streams*).
 - Ces structures contiennent notamment un tampon (*buffer*) permettant d'optimiser les opérations d'entrées/sorties.
- ▶ On y retrouve les équivalents des fonctions qu'on a vues :
 - `FILE *fopen(const char *restrict pathname, const char *restrict mode);`
 - `int fclose(FILE *stream);`
 - `size_t fread(void ptr[restrict .size * .nmemb], size_t size, size_t nmemb, FILE *restrict stream);`
 - `size_t fwrite(const void ptr[restrict .size * .nmemb], size_t size, size_t nmemb, FILE *restrict stream);`
 - `off_t fseeko(FILE *stream, off_t offset, int whence);`

Changement de niveau

- ▶ La bibliothèque **stdio** permet aussi de “changer de niveau” si besoin.
- ▶ **fdopen** : créer un flux à partir d'un descripteur de fichier
 - `FILE *fdopen(int fd, const char *mode);`
- ▶ **fileno** : récupérer le descripteur de fichier d'un flux
 - `int fileno(FILE *stream);`
- ▶ Il est fortement déconseillé de mélanger l'utilisation de fonctions de haut et bas niveau sur un même fichier (à cause des tampons des **FILE**).
- ▶ La bibliothèque **stdio** crée automatiquement des **FILE *** pour les trois descripteurs de fichiers standards et les assigne à des variables globales.
 - `stdin` est `fdopen(0)`,
 - `stdout` est `fdopen(1)`,
 - `stderr` est `fdopen(2)`,

Liens et renommage

- ▶ `#include <unistd.h>`
- ▶ `link`: créer un nouveau nom pour un fichier
 - `int link(const char *oldpath, const char *newpath);`
- ▶ `symlink`: créer un nouveau nom symbolique pour un fichier
 - `int symlink(const char *target, const char *linkpath);`
- ▶ `readlink`: récupérer la valeur d'un lien symbolique
 - `ssize_t readlink(const char *restrict pathname, char *restrict buf, size_t bufsiz);`
- ▶ `unlink`: supprimer un nom et possiblement le fichier associé
 - `int unlink(const char *pathname);`
- ▶ `rename`: renommer ou déplacer un fichier
 - `#include <stdio.h>`
 - `int rename(const char *oldpath, const char *newpath);`

► **access** : vérifier les permissions de l'utilisateur·ice sur un fichier

- `#include <unistd.h>`
- `int access(const char *pathname, int mode);`

► **stat** : récupérer les informations sur un fichier

- `#include <sys/stat.h>`
- `int stat(const char *restrict pathname, struct stat *restrict statbuf);`

► **chmod** : changer les permissions d'un fichier

- `#include <sys/stat.h>`
- `int chmod(const char *pathname, mode_t mode);`

► **chown** : changer le propriétaire d'un fichier

- `#include <unistd.h>`
- `int chown(const char *pathname, uid_t owner, gid_t group);`

► **umask** : changer le masque du mode de création de fichier

- `#include <sys/stat.h>`
- `mode_t umask(mode_t mask);`

Répertoires

► **getcwd** : récupérer le répertoire de travail

- `#include <unistd.h>`
- `char *getcwd(char buf[.size], size_t size);`

► **chdir** : changer le répertoire de travail

- `#include <unistd.h>`
- `int chdir(const char *path);`

► **mkdir** : créer un répertoire

- `#include <sys/stat.h>`
- `int mkdir(const char *pathname, mode_t mode);`

► **rmdir** : supprimer un répertoire

- `#include <unistd.h>`
- `int rmdir(const char *pathname);`

► Il existe également des fonctions de haut niveau facilitant la manipulation de répertoires par flux de type **DIR *** et de structures **dirent** :

- **opendir** : ouvrir un flux de répertoire
- **closedir** : fermer un flux de répertoire
- **readdir** : lire la prochaine entrée **dirent** dans un flux de répertoire