[SY5]

“

Wiesław Zielonka
zielonka@liafa.univ-paris-diderot.fr
www.liafa.univ-paris-diderot.fr/~zielonka

1. Fichiers
    1. Systèmes de fichiers
    2. Types de fichiers
    3. Chemin absolu
    4. Chemin relatif
    5. Le répertoire courant d’un processus
    6. Changer le répertoire courant dans un programme C
    7. Les répertoires
    8. Les répertoires
    9. Caractéristiques de fichiers
    10. Type de fichier (programme C)
    11. Les droits d'accès
        1. Les fichiers ordinaires:
        2. les répertoires
    12. Les droits d'accès
        1. Macro-constantes:
    13. Identité de fichier
    14. Parcours de répertoire
    15. Création et suppression d'un répertoire
    16. Liens durs
    17. Création de liens durs avec link
    18. Exemple link
    19. Suppression de lien dur
        1. Renommage de lien dur
    20. Gestion d'erreurs en C

Fichiers

1. Systèmes de fichiers

- linux : ext3fs, ext4fs, ReiserFS
- MSWindows : FAT16, FAT32, NTFS
- CD-ROM : ISO 9660
- Mac OS : HFS+

Différents systèmes de fichiers peuvent exister sur le même disque (par exemple NTFS et ext4fs si windows et linux installés sur le même DD). Chaque système de fichiers sur le DD install sur sa propre partition (volume).

D'un point de vue de processus : le système principal installé à la racine (/), et d'autres systèmes **"montés"** et visibles comme de **sous-répertoires**, commandes ``mount et umount````.

2. Types de fichiers

- fichiers ordinaires - suite d'octets
- les répertoires,
- fichiers spéciaux FIFO (pipe),
- fichiers spéciaux bloc,
- fichiers spéciaux caractères,
- liens symboliques,
- les sockets.

3. Chemin absolu

Le chemin absolu permet de situer un fichier depuis la racine, le chemin absolu commence par `/`

```
/home/tom/enseignement/Systemes/cours2.pdf
/home/tom/enseignement/../../mike/./photos/rome.jpeg
```

- `..` permet de remonter vers le répertoire père.
- `.` d´esigne le r´epertoire courant.

> *A noter que le père de* `/` *est* `/.`

4. Chemin relatif

c'est le chemin qui ne commence pas par `/`

```
enseignement/Systemes/cours1.tex
enseignement/../../mike/photos/paris.jpeg
```

Mais ce chemin est relatif vis à vis de quoi? Relatif par rapport au **répertoire** courant du processus. Pour connaître le répertoire courant de shell qui contrôle le terminal exécutez la commande UNIX `pwd` . Pour changer le répertoire courant de shell la commande UNIX `cd` .

5. Le répertoire courant d'un processus

Pour récupérer le chemin absolu vers le répertoire courant de processus depuis un programme C

```
char* getcwd (char* buf, size_t taille_buf)
```

le paramètre buf le pointeur vers le tampon où `getcwd` place le résultat, `taille_buf` la taille de tampon. Si la taille de tampon trop petite alors `getcwd` retourne `NULL` et `errno==ERANGE` . dans ce cas il convient de r´eessayer avec le tampon plus grand.

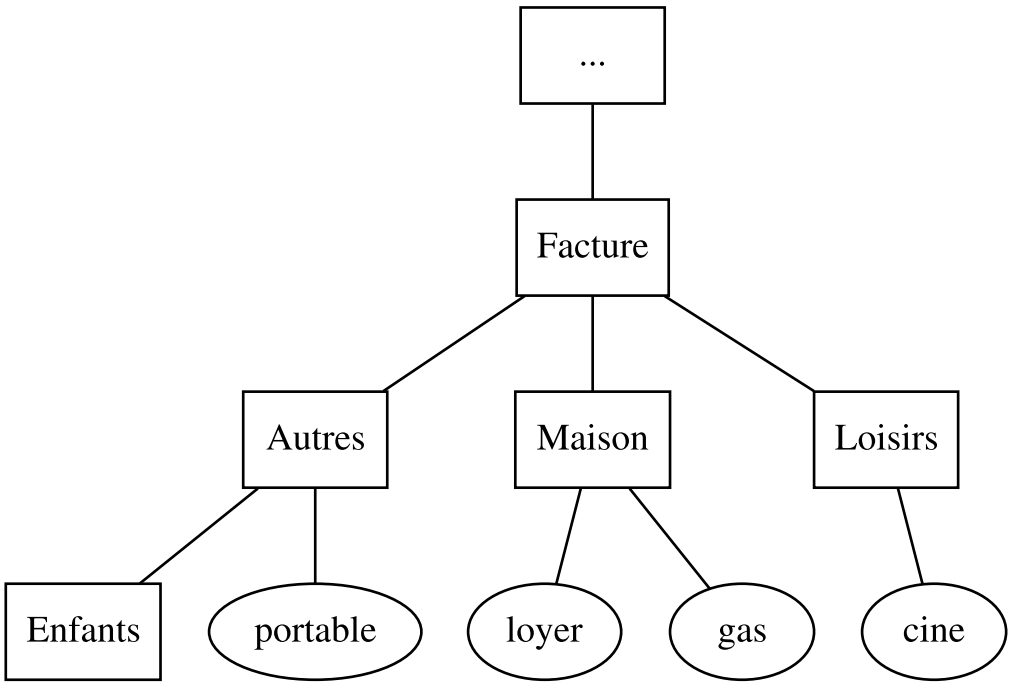
```
char *tampon, * path;
size_t i = 64;
for ( ; ; ) {
    tampon = malloc(i) ;
if(tampon == NULL){/* gerer erreur de malloc() */ }
    path = getcwd(tampon, i);
    if(path == NULL && errno == ERANGE){
        i *= 2;
        free(tampon);
        continue ;
    }
    else if(path == NULL){/* d'autres problemes*/
        free(tampon);
        break
    }
}
```

6. Changer le répertoire courant dans un programme C

```
int chdir(constchar* chemin);
```

`chemin` :**chemin absolu ou relatif** vers le nouveau répertoire courant.

7. Les répertoires



8. Les répertoires

Le répertoire c'est un fichier spécial qui contient la liste de noms de fichiers (les fichiers qui se trouvent dans le répertoire) avec les liens vers les fichiers correspondant sur le disque. Dans la liste de noms deux noms particuliers : `.` *(dot) avec le lien vers le répertoire lui-même,* `..` (dot dot) avec le lien vers le répertoire père.

9. Caractéristiques de fichiers

```
struct stat {
    dev_t st_dev; /*identificateur de systeme de fichiers */
    ino_t st_ino; /*matricule du fichier (numero inode) */
    uid_t st_uid; /*identificateur du proprietaire */
    gid_t st_gid; /*identificateur du groupe proprietaire */
    mode_t st_mode; /* typeetmodedufichier */
    nlink_t st_nlink; /* nombre de liens durs sur le fichier */
    off_t st_size; /
* taille de fichier en octets pour le fichier ordinaire */
    time_t st_atime; /* temps universel du dernier acces */
    time_t st_mtime; /
* temps universel de la derniere modification de donnees */
    time_t st_ctime; /
* temps universel de la derniere modification de caracteristiques */
}
```

```
int stat(const char* chemin, struct stat* st)
int lstat(const char* chemin, struct stat* st)
```

La différence entre `stat()` et `lstat()` concerne le traitement de liens symboliques.

10. Type de fichier (programme C)

macro-fonction	**type de fichier**	**type affiché par `ls -l`**
S_ISREG(tamp.st_mode)	fichier régulier	-
S_ISFIFO(tamp.st_mode)	fichier spécial FIFO (tube)	p
S_ISCHR(tamp.st_mode)	type spécial caractère	c
S_ISBLK(tamp.st_mode)	type spécial bloc	b
S_ISDIR(tamp.st_mode)	type spécial répertoire	d
S_ISLNK(tamp.st_mode)	lien symbolique	l
S_ISSOCK(tamp.st_mode)	socket	s

```
#define _POSIX_C_SOURCE 200112L
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
struct stat tamp;
if(lstat("toto", &tamp) < 0){
    /* gerer les erreurs de lstat */
}
if(S_ISREG(tamp.stmode)){ /* fichier regulier */
else if(SISFIFO(tamp.stmode)){ /* FIFO */}
else if(SISCHR(tamp.stmode)){ /* special type */
else if(SISBLK(tamp.stmode)){ /* special type */
else if(SISDIR(tamp.stmode)){ /* repertoire*/
/*etc*/
```

11. Les droits d'accès

11.1. Les fichiers ordinaires:

`r`	lecture
`w`	écriture
`x`	exécution

11.2. les répertoires

`r`	lecture
`w`	écriture (création ou suppression d'une entrée dans le répertoire)
`x`	passage

Pour lire un fichier `f` (y inclus pour la lecture d'un répertoire `f`) il faut avoir les droits de passage sur tous les répertoires depuis la racine jusqu'au père de `f` et les droits de lecture sur `f` .

12. Les droits d'accès

12.1. Macro-constantes:

`S_IRUSR`	r propriétaire
`S_IWUSR`	w propriétaire
`S_IXUSR`	x propriétaire
`S_IRGRP`	r groupe
`S_IWGRP`	w groupe
`S_IXGRP`	x groupe
`S_IROTH`	r les autres
`S_IWOTH`	w les autres
`S_IXOTH`	x les autres

```
struct stat tamp;
if(lstat("toto",&tamp) < 0 ){
    /* gerer les erreurs de lstat */
}
if(tamp.stmode & SIRUSR){
    /* proprietaire posede le droit de lecture sur toto */
}
```

13. Identité de fichier

L'identifiant de système de fichier `st_dev` identifie de façon unique le système de fichier. Le numéro de inode (`st_ino`) est unique pour chaque fichier dans le même système de fichier, par contre les fichiers dans les systèmes de fichiers différents peuvent avoir les numéros de inode identique.

```
char *chemin1, *chemin2 ;
struct stat buf1, buf2;
...
stat(chemin1, &buf1);
stat(chemin2, &buf2);
if(buf1.stdev == buf2.stdev && buf1.stino == buf2.stino){
    /*chemin1 et chemin2 menent vers le meme fichier */
}
```

14. Parcours de répertoire

```
#include <dirent.h>
DIR *opendir(const char *pathname);
struct dirent *readdir(DIR *dp);
void rewinddir(DIR *dp);
int closedir(DIR *dp);
```

1. Ouvrir le répertoire avec `opendir()` ,
2. lire dans la boucle avec `readdir()` , chaque lecture donne une nouvelle entrée, y inclus dot et dot dot, à la fin de parcours `readdir()` retourne `NULL` ,
3. après le parcours fermer avec `closedir()` `rewinddir()` permet de revenir au début de répertoire.

```
struct dirent {
    ino_t d_ino; /* i-node number */
    char d_name[NAME_MAX + 1]; /* null-terminated */
}
```

La valeur de `NAME_MAX` peut être récupérée avec un appel à `fpathconf` .

```
DIR *flot;
struct dirent *entree;
/* ouvrir repertoire */
if((flot=opendir(nomrep)) == NULL){
    perror("opendir");
    exit(0);
}
errno = 0;
for(;;){
    entree = readdir(fl_t); /*lire une entree de repertoire */
    if(entree==NULL){
        if(errno){
            /* gerer erreur de parcours */
        }else{ /* fin de parcours */
            break;
        }
    }
    if(strcmp(entree->d_name, ".")==0 || strcmp(entree-
>d_name, "..") == 0)
        continue;
    printf("%s/%s\n",nom_rep, entree->d_name);
    }
    closedir ( flot );
```

15. Création et suppression d'un répertoire

```
int mkdir(const char *chemin, mode_t mode);
```

```
mkdir("Enseignements/
Systemes1", S IRUSR | S IWUSR | S IXUSR | S IXGRP);
```

Le deuxième paramètre indique les droits demandés (mis pas forcement accordés).

```
int rmdir(const char *chemin);
```

16. Liens durs

Liens durs sont créés à la création d'un fichier `open()`, `creat()` ou d'un répertoire `mkdir()`.

17. Création de liens durs avec link

Mais on peut créer des liens durs additionnels vers les fichiers existants avec : la commande UNIX :

```
ln source dest
```

ou la fonction POSIX :

```
int link(const char *source, const *dest)
```

qui crée un lien dur `dest` qui pointe vers le même fichier que source.

18. Exemple link

Le répertoire curant : Factures

```
link("Autres/portable", "Loisir/portable_Marc") ;
```

19. Suppression de lien dur

```
int un link(const char *chemin)
```

supprime le lien dur. Le fichier lui même sera supprimé par le système d'exploitation si le compteur de liens dur pour ce fichier passe à 0 et si le fichier n'est pas ouvert par un processus quelconque. **Il n'y a pas de fonction de suppression de fichier mais uniquement la fonction de suppression de liens durs.**

```
unlink("/home/marion/Systemes/cours3.tex" )
```

réussit si le processus possède le droit de passages sur: `/, /home, /home/marion, /home/marion/Systemes` et le droit d'écriture sur `/home/marion/Systemes`. Aucun droit sur cours3.tex n'est exigé.

19.1. Renommage de lien dur

```
int rename(const char *ancien, constchar *nouveau)
```

`rename("Maison/loyer", "Autre/loyeraregler")` supprime le lien dans Maison et crée un lien dans Autres. Quels droits faut-il avoir pour effectuer cette manipulation?

20. Gestion d'erreurs en C

Les appels système signalent une erreur en retournant soit une valeur inférieure à `0` soit `NULL`. Le numéro d'erreur est mis dans la variable globale `errno` déclarée dans le fichier `errno.h` qu'il convient d'inclure avec la directive `include`.

errno n'est pas automatiquement remise à 0 après une erreur.

```
void perror(constchar *str)
```

La fonction `perror()` affiche sur la sortie d'erreurs standard le message d'erreur qui correspond à la valeur courante de errno précédé par la chaîne de caractères str passée en argument.

```
char *strerror(int num)
```

retourne le message d'erreur qui correspond au numéro d'erreur `num`.

Dans mes programmes j'utiliserai la macro-fonction suivante pour gérer les message d'erreur. `__FILE__` est une macro dont la valeur est le nom de fichier source, `__LINE__` est une macro dont la valeur est le numéro de la ligne dans le fichier source.

```
/* ***************** panic.h ******************* */
#ifndef PANIC_H
#define PANIC_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#define PANIC(fin)
    do{
        fprintf(stderr, "\n error in file %s in line %d: %s
\n", __FILE__,__LINE__,strerror(errno));
        if(fin > 0) exit(fin);
        errno=0;\
    }while(0)
#endif
```