répertoires

Bien qu'un répertoire ne soit qu'un fichier comme un autre à un niveau relativement bas, son contenu est sensible, toute altération brisant le formatage de ses données mets en péril la cohérence du système de nommage.
C'est pourquoi la manipulation de leur contenu ne peut être réalisée que par des fonctions particulières qui garantissent la bonne tenue de la cohérence.

ouverture d'un répertoire : `opendir`

La consultation d'un répertoire s'effectue *via* un flot de répertoire dont le type est `DIR`. Comme `FILE` il s'agit d'un type opaque représentant un itérateur sur le répertoire. Un tel objet est obtenu par appel à :

Definition (`opendir`)

```
#include <dirent.h>
DIR *opendir(const char *référence);
```

où la *rfrence* doit désigner un fichier de type répertoire.
En cas de succès une adresse valide de `DIR` est obtenue et le flot pointe alors vers la première entrée, et `NULL` sinon.

itération sur un répertoire : `readdir`

L'obtention d'une entrée de répertoire (et le pointage sur l'entrée suivante) s'effectue *via* :

Definition (`readdir`)

```
#include <dirent.h>
struct dirent *readdir(DIR *flot);
```

En cas de succès, cette fonction renvoie l'adresse d'une structure décrivant l'entrée courante du flot et déplace le flot sur la structure suivante. En cas d'échec ou de fin de flot, `NULL` est renvoyé (si c'est la fin de flot a variable `errno` n'est pas modifiée).

entrée de répertoire : `struct dirent`

Definition (`struct dirent`)

Le seul membre normalisé de cette structure est le champ **`char`** `d_name[]` qui contient le nom d'un lien contenu dans le répertoire (on rappelle qu'un répertoire est essentiellement une liste de liens).

fermeture d'un répertoire : `closedir`

Definition (`closedir`)

```
#include <dirent.h>
int closedir(DIR *flot);
```

Cette fonction permet d'obtenir la libération du *flot*. En cas de succès 0 est renvoyé et -1 (+`errno`) en cas d'échec.

Exemple

```
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h>

void ls(const char *name) {
  DIR *d = opendir(name);
  if (d==NULL) return;
  printf("Repertoire␣%s---------------\n",name);
  struct dirent *e;
  while ( (e=readdir(d)) != NULL)
    puts(e->d_name);
  closedir(d);
}

int main(int argc,char *argv[]) {
  if (argc==1) ls(".");
  else while (--argc) ls (*++argv);
  exit(0);
}
```

```
$ ./mylittlels /tmp/toto /tmp/titi
Repertoire /tmp/toto---------------
.
..
.config
truc
Repertoire /tmp/titi---------------
.
..
$
```

L'itération fait apparaître **tous** les liens (y compris `.` et `..`).

consultation des propriétés d'un i-nœud : `stat`

On a déjà dit qu'à un i-nœud est associé non seulement un contenu mais aussi des méta-données. La fonction :

Definition (`stat`)

```
#include <sys/stat.h>
int stat(const char *nom,struct stat *s);
```

permet d'obtenir dans *s* les valeurs des méta-données de l'i-nœud désigné par son *nom*. En cas de succès la fonction renvoie 0 et -1 sinon.

struct stat

Definition (struct stat)

Une **struct** stat contient au moins les membres suivants (leur ordre n'est pas spécifié) :

```
struct stat {
  mode_t st_mode;
  ino_t st_ino;
  dev_t st_dev;
  nlink_t st_nlink;
  uid_t st_uid;
  gid_t st_gid;
  off_t st_size;
  time_t st_atime;
  time_t st_mtime;
  time_t st_ctime;
};
```


st_mode

Ce champ décrit (entre autres) l'accessibilité du fichier, c'est-à-dire définit pour l'i-nœud quels processus sont autorisés à faire quoi de l'i-nœud. Il est (en partie) une combinaison des valeurs symboliques suivantes (masque de bits) :

S_IRUSR, S_IWUSR, S_IXUSR qui représentent respectivement les droits de lecture, écriture et exécution pour un processus dont le propriétaire est aussi propriétaire de l'i-nœud;

S_IRGRP, S_IWGRP, S_IXGRP qui représentent les droits pour un processus dont le propriétaire n'est pas propriétaire de l'i-nœud mais dont le groupe propriétaire est aussi le groupe propriétaire de l'i-nœud;

S_IROTH, S_IWOTH, S_IXOTH qui représentent les droits pour les processus qui n'entrent dans aucun des deux cas précédents.

Pour des raisons pratiques sont aussi définies les valeurs
`S_IRWXU=S_IRUSR | S_IWUSR | S_IXUSR`,
`S_IRWXG=S_IRGRP | S_IWGRP | S_IXGRP` et
`S_IRWXO=S_IROTH | S_IWOTH | S_IXOTH`.
Sont aussi disponibles les valeurs :

- `S_ISUID` qui représente le set-uid bit et qui permet d'obtenir à l'exécution d'un code l'outre-passement d'identité;
- `S_ISGID` qui représente le set-gid bit et qui permet d'obtenir à l'exécution d'un code l'outre-passement d'identité;
- `S_ISVTX` qui représente le sticky-bit

Ces valeurs sont particulières et feront l'objet de détails dans un cours de systèmes avancé.

Exemple

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>

#define SHOW(s,v,c) (((s).st_mode&(v))?(c):'-')

void ls(const char *name) {
  struct stat s;
  if ( stat(name,&s)!=0 ) return;
  printf("%c%c%c%c%c%c%c%c%c␣%s\n",
         SHOW(s,S_IRUSR,'r'), SHOW(s,S_IWUSR,'w'),
         SHOW(s,S_IXUSR,'x'),
         SHOW(s,S_IRGRP,'r'), SHOW(s,S_IWGRP,'w'),
         SHOW(s,S_IXGRP,'x'),
         SHOW(s,S_IROTH,'r'), SHOW(s,S_IWOTH,'w'),
         SHOW(s,S_IXOTH,'x'),
         name);
}

int main(int argc,char *argv[]) {
  if (argc==1) ls(".");
  else while (--argc) ls(*++argv);
  exit(0);
}
```

```
$ ls -al /tmp/
total 48
drwxrwxrwt  13 root   wheel    442 20 oct 14:28 .
drwxr-xr-x@  6 root   wheel    204 12 oct  2015 ..
-rw-rw-rw-   1 yunes  wheel      0 20 oct 06:11 .keystone_install_lock
drwx------   3 yunes  wheel    102 20 oct 06:11 com.apple.launchd.QP6VPFJXwV
drwx------   3 yunes  wheel    102 20 oct 06:11 com.apple.launchd.Tns8H0BePX
drwx------   3 yunes  wheel    102 20 oct 06:11 com.apple.launchd.VzpdUyrgYo
-rw-r--r--@  1 yunes  wheel      0 20 oct 06:11 ct.shutdown
drwxr-xr-x   3 root   wheel    102  2 oct 02:39 noticeboard
-rwxrwxrwx@  1 root   wheel     29 20 oct 06:11 sperion
-rwxr-xr-x   1 yunes  wheel  15724 20 oct 12:22 t
-rw-r--r--   1 yunes  wheel    418 20 oct 12:22 t.cpp
drwxr-xr-x   2 yunes  wheel     68 20 oct 14:28 titi
drwxr-xr-x   4 yunes  wheel    136 20 oct 14:28 toto
$ ./mylsacces /tmp/*
rwx------ /tmp/com.apple.launchd.QP6VPFJXwV
rwx------ /tmp/com.apple.launchd.Tns8H0BePX
rwx------ /tmp/com.apple.launchd.VzpdUyrgYo
rw-r--r-- /tmp/ct.shutdown
rwxr-xr-x /tmp/noticeboard
rwxrwxrwx /tmp/sperion
rwxr-xr-x /tmp/t
rw-r--r-- /tmp/t.cpp
rwxr-xr-x /tmp/titi
rwxr-xr-x /tmp/toto
$
```


droits d'accès le retour

Il ne faut pas se méprendre sur la sémantique des droits d'accès.
Pour un **fichier de type tout sauf répertoire et lien symbolique**, le droit de lecture signifie l'autorisation de lire le contenu, le droit d'écriture signifie l'autorisation de modifier le contenu (y compris effacer le contenu!), le droit d'exécution ne sert que pour les fichiers qui contiennent du code (exécutable ou script) et définit l'autorisation de demander l'exécution du code correspondant.

Pour les **répertoires** la situation est quelque peu différente, en tous cas il faut faire un effort d'interprétation. Le droit de lecture correspond à la possibilité de lire le contenu du répertoire, donc connaître la liste des liens. Le droit d'écriture correspond à la possibilité de modifier le contenu du répertoire, donc modifier la liste des liens, par exemple supprimer un lien (la suppression n'est pas liée à une propriété du fichier mais au répertoire qui le contient!). Le droit d'exécution correspond lui à la possibilité de faire apparaître ce répertoire dans un chemin, c'est le droit de traverser.
Il existe d'autres possibilités qui utilisent les set-uid, set-gid et sticky bits, mais nous laisserons cela de côté pour ce cours.

st_mode

Ce champ décrit aussi le type de l'i-nœud. Il est déterminé par l'intermédiaire des macro-définitions suivantes (celles du standard et pas toutes) :

`S_ISDIR(mode)` qui renvoie vrai si l'i-nœud est de type répertoire, faux sinon;

`S_ISREG(mode)` qui renvoie vrai si l'i-nœud est de type fichier ordinaire, faux sinon;

`S_ISCHR(mode)` qui renvoie vrai si l'i-nœud est un périphérique en mode caractère;

`S_ISBLK(mode)` qui renvoie vrai si l'i-nœud est un périphérique en mode bloc;

`S_ISFIFO(mode)` qui renvoie vrai si l'i-nœud est un tube;

`S_ISLNK(mode)` qui renvoie vrai si l'i-nœud est un lien symbolique;

Exemple

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>

#define SHOW(s,v,c) (((s).st_mode&(v))?(c):'-')
#define TYPE(s) (                                   \
                 S_ISDIR((s).st_mode)?'d':          \
                 (S_ISREG((s).st_mode)?'-':         \
                  (S_ISCHR((s).st_mode)?'c':        \
                   (S_ISBLK((s).st_mode)?'b':       \
                    (S_ISFIFO((s).st_mode)?'p':     \
                     (S_ISLNK((s).st_mode)?'l':     \
                      '?'                           \
                     )                              \
                    )                               \
                   )                                \
                  )                                 \
                 )                                  \
                                                    )

void ls(const char *name) {
  struct stat s;
  if ( stat(name,&s)!=0 ) return;
  printf("%c%c%c%c%c%c%c%c%c%c␣%s\n",
         TYPE(s),
         SHOW(s,S_IRUSR,'r'), SHOW(s,S_IWUSR,'w'),
         SHOW(s,S_IXUSR,'x'),
         SHOW(s,S_IRGRP,'r'), SHOW(s,S_IWGRP,'w'),
         SHOW(s,S_IXGRP,'x'),
         SHOW(s,S_IROTH,'r'), SHOW(s,S_IWOTH,'w'),
         SHOW(s,S_IXOTH,'x'),
         name);
}

int main(int argc,char *argv[]) {
  if (argc==1) ls(".");
  else while (--argc) ls(*++argv);
  exit(0);
}
```



```
$ ./mylsacces2 /tmp/*
drwx------ /tmp/com.apple.launchd.QP6VPFJXwV
drwx------ /tmp/com.apple.launchd.Tns8H0BePX
drwx------ /tmp/com.apple.launchd.VzpdUyrgYo
-rw-r--r-- /tmp/ct.shutdown
drwxr-xr-x /tmp/noticeboard
-rwxrwxrwx /tmp/sperion
-rwxr-xr-x /tmp/t
-rw-r--r-- /tmp/t.cpp
drwxr-xr-x /tmp/titi
drwxr-xr-x /tmp/toto
$
```

st_ino, st_dev

Ces deux informations ne sont quasiment pas utiles, mais on essentiellement un usage interne.
Toufefois, le couple de ces valeurs identifient de façon unique un «fichier» dans une arborescence. La valeur de `st_dev` peut-être différente entre deux incarnations du système (démarrages).
La valeur `st_ino`, seule, identifie de façon unique un «fichier» dans un système de fichiers, cette valeur est permanente.

st_nlink

Il s'agit du nombre de liens physiques qui désignent cet i-nœud. L'i-nœud ne sera libéré que si (condition nécessaire mais non suffisante) ce nombre tombe à zéro.

st_uid, st_gid

Il s'agit des identifiants du propriétaire et du groupe propriétaire. Ces identifiants sont utilisés pour déterminer les accès à l'i-nœud.

st_size

Il s'agit de la taille, exprimée en octets, du contenu de l'i-nœud pour tous les i-nœuds de type ordinaires (regular) ou liens symboliques (symbolic link). Pour les autres types, cette valeur n'est pas spécifiée.

Exemple

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>

void ls(const char *name) {
  struct stat s;
  if ( stat(name,&s)!=0 ) return;
  printf("%8jd␣%s\n",(intmax_t)s.st_size,
     name);
}

int main(int argc,char *argv[]) {
  if (argc==1) ls(".");
  else while (--argc) ls(*++argv);
  exit(0);
}
```



```
$ ./mylstaille /tmp/*
     102 /tmp/com.apple.launchd.QP6VPFJXwV
     102 /tmp/com.apple.launchd.Tns8H0BePX
     102 /tmp/com.apple.launchd.VzpdUyrgYo
       0 /tmp/ct.shutdown
     102 /tmp/noticeboard
      29 /tmp/sperion
   15724 /tmp/t
     418 /tmp/t.cpp
      68 /tmp/titi
     136 /tmp/toto
 1619009 /tmp/wifi-a1vint.log
$
```

dates

À chaque i-nœud sont associées trois horodatages :

- `st_atime` qui représente la date de dernier accès en lecture au contenu;
- `st_ctime` qui représente la date de dernière modification l'i-nœud;
- `st_mtime` qui représente la date de dernière modification (écriture) du contenu.

Sur certains systèmes (comme DARWIN), il existe une quatrième date, celle de création du nœud.
Ces données sont fournies dans le type `time_t`.

`time_t`

`time_t` est un type permettant de représenter une date exprimée en secondes depuis l'`EPOCH` (l'origine des temps POSIX), c'est-à-dire le nombre de secondes écoulées depuis le 1er janvier 1970, 0 heures, 0 minutes, 0 secondes en temps coordonné universel.
Sa conversion en une chaîne de caractères peut être obtenue *via* :

Definition (`ctime`)

```
#include <time.h>
char *ctime(const time_t *timer);
```

qui renvoie une chaîne de 25 caractères de long dont le dernier est un retour à la ligne et qui représente une écriture de la date exprimée par l'argument.

Exemple

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

void ls(const char *name) {
  struct stat s;
  if ( stat(name,&s)!=0 ) return;
  char at[26], mt[26], ct[26];
  strncpy(at,ctime(&(s.st_atime)),26);
  at[strlen(at)-1] = '\0';
  strncpy(mt,ctime(&(s.st_mtime)),26);
  mt[strlen(mt)-1] = '\0';
  strncpy(ct,ctime(&(s.st_ctime)),26);
  ct[strlen(ct)-1] = '\0';
  printf("%s %s %s %s\n",at,mt,ct,name);
}

int main(int argc,char *argv[]) {
  if (argc==1) ls(".");
  else while (--argc) ls(*++argv);
  exit(0);
}
```



```
$ ./mylsdate zzzzz.c
Fri Oct 21 15:49:18 2016 Fri Oct 21 15:49:12
    2016 Fri Oct 21 16:18:27 2016 zzzzz.c
$
```

fonctions de création d'i-nœuds

Comme il existe des i-nœuds de différents types, il existe pour chaque type (ou presque) une fonction de création particulière :

- `creat` afin de créer un fichier ordinaire (on verra que cette fonction est rarement appelée);
- `mkdir` afin de créer un répertoire;
- `mkfifo` afin de créer un tube nommé;
- `mknod` qui est la fonction générique de création d'i-nœud (nous ne l'étudierons pas).

masque de création : `umask`

Avant toute chose, précisons comment les droits d'accès sont contrôlés à la création d'un i-nœud.
Le contrôle final du positionnement des droits d'accès lors de la création d'un i-nœud est laissé à l'utilisateur. Pour cela, chaque processus possède un **masque courant** (*user mask*) permettant d'affaiblir les droits établis a-priori par une application. Tout les droits positionnés dans le masque courant correspondent à l'affaiblissement souhaité (ce sont donc des droits qui ne seront pas positionnés).

Definition (`umask`)

La fonction :

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

permet d'obtenir le positionnement du nouveau *masque* courant. La valeur de retour est la valeur du masque avant sa modification.

création d'un répertoire : `mkdir`

Definition (`mkdir`)

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

permet d'obtenir (à condition que les droits nécessaires soient obtenus) la création du répertoire de *chemin* indiqué, et avec les *droits* d'accès (restreints par le masque courant - voir page 739).
En cas de succès la fonction renvoie 0 et -1 sinon (consulter `errno` pour les raisons de l'échec).

Exemple

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>

int main(int argc,char *argv[]) {
  if (argc!=2) exit(1);
  if (mkdir(argv[1],S_IRWXU | S_IRWXG |
     S_IRWXO)==-1)
    perror(argv[0]),exit(1);
  exit(0);
}
```



```
$ umask
022
$ ./mymkdir AAA
$ ls -ld AAA
drwxr-xr-x  2 yunes  staff  68 24 oct 15:50
   AAA
$ umask 000
$ rmdir AAA
$ ./mymkdir AAA
$ ls -ld AAA
drwxrwxrwx  2 yunes  staff  68 24 oct 15:51
   AAA
$ umask 111
$ rmdir AAA
$ ./mymkdir AAA
$ ls -ld AAA
drw-rw-rw-  2 yunes  staff  68 24 oct 15:51
   AAA
$
```

mkfifo

Definition (mkfifo)

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

permet de créer un i-nœud de type tube nommé.
La fonction renvoie 0 en cas de succès, -1 sinon et `errno`.
Pour plus de détails sur l'usage des tubes (nommés ou non, voir page **??**).

creat

Definition (creat)

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

permet de créer un i-nœud de type ordinaire, mais elle n'est que très rarement employée. On lui préfère la fonction `open`, et d'ailleurs un appel à `creat(path,mode)` fonctionne comme un appel à `open(path, O_WRONLY|O_CREAT|O_TRUNC, mode)` (voir page **??**).
La fonction renvoie >0 en cas de succès, -1 sinon et `errno`.

link

Definition (`link`)

```
#include <unistd.h>
int link(const char *path1, const char *
    path2);
```

permet d'obtenir (si les droits sont suffisants comme de bien entendu), un nouveau lien *path*2 qui désigne le même i-nœud que *path*1 désigne.
En cas de succès la fonction renvoie 0 et -1 sinon (+`errno`).

⚠ Attention

Cette fonction ne permet pas à un utilisateur normal de créer un lien physique sur un répertoire...

«suppression de fichiers»

Nous l'avons déjà dit il n'existe pas de fonction qui supprime (libère) un i-nœud. Un tel effet n'est obtenu que comme effet secondaire (éventuellement) de la suppression d'un lien. En effet, lorsqu'on supprime un lien, cela n'a pour effet (outre la suppression de l'entrée de répertoire correspondante) que de décrémenter le compteur de liens de l'i-nœud.
C'est lorsque ce compteur est égal à 0 (donc qu'il ne possède plus de lien, donc plus de «nom») que le système procédera (après un certain temps) à sa suppression (puisqu'il n'a plus de nom, on ne peut plus explicitement y accéder par une «ouverture»).
Il s'agit d'un **ramasse-miettes** (*garbage collector*).

suppression de lien : `unlink`

Definition (`unlink`)

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

permet d'obtenir la suppression d'une entrée de répertoire (sous réserve des droits afférents) et donc de décrémenter le compteur de liens de l'i-nœud qui correspond.
En cas de succès cette fonction renvoie 0, -1 sinon (+`errno`).

Exemple

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc,char *argv[]) {
  if (argc<2) exit(1);
  int flag = 1;
  while (--argc)
    if (unlink(*++argv)==-1)
      perror(argv[0]);
    else flag = 0;
  exit(flag);
}
```

suppression de répertoire ; `rmdir`

Definition (`rmdir`)

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

permet d'obtenir la suppression d'un répertoire (sous réserve des droits afférents et qu'il soit «vide»).
En cas de succès cette fonction renvoie 0, -1 sinon (+`errno`).

renommage

Cette fonction est la même qu'en C : `rename`, voir page 576. Elle permet la modification du nom d'un lien et/ou son déplacement.

⚠ Attention

Il est possible que le renommage provoque la recopie du fichier (si les deux noms ne sont pas dans le même système de fichiers) mais ce fonctionnement n'est pas garanti.

modification des propriétés

À chaque i-nœud sont associés des propriétés, lesquelles peuvent être modifiées implicitement (la taille par exemple) ou explicitement.

modification des droits : `chmod`

Definition (`chmod`)

La fonction :

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

permet d'obtenir le positionnement à la valeur *mode* des droits d'accès (étendus) de l'i-nœud correspondant au chemin (sous réserve d'en avoir le droit).
Les droits d'accès sont une combinaison des droits de base décrits pour `st_mode`, voir page 719.
En cas de succès la fonction renvoie 0, -1 sinon (+`errno`).

modification des propriétaires : chown

Definition (chown)

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

permet d'obtenir le positionnement du propriétaire et du groupe propriétaire (sous réserve d'en avoir les droits).
En cas de succès la fonction renvoie 0, -1 sinon (+errno).

fonctions sur les utilisateurs et groupes

Il peut être utile de passer d'un nom à l'identité d'un utilisateur ou vice-versa. C'est le rôle des fonctions :

Definition (getpwuid, getpwnam)

```
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
struct passwd *getpwnam(const char *name);
#include <grp.h>
struct group *getgrgid(gid_t gid);
struct group *getgrnam(const char *name);
```

struct passwd

Definition (struct passwd)

La structure `passwd` doit contenir au moins les champs suivants :

`char *pw_name` le nom de l'utilisateur;

`uid_t pw_uid` l'identification de l'utilisateur;

`gid_t pw_gid` l'identification du groupe;

`char *pw_dir` le répertoire privé de l'utilisateur;

`char *pw_shell` le shell par défaut.

struct group

Definition (struct group)

La structure `group` doit contenir au moins les champs suivants :

`char *gr_name` le nom du groupe;

`gid_t gr_gid` le numéro du groupe;

`char **gr_mem` la liste des membres du groupe (terminée par un pointeur nul).

Exemple

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <pwd.h>
#include <grp.h>

void ls(const char *name) {
  struct stat s;
  if ( stat(name,&s)!=0 ) return;
  struct passwd *p;
  char *uidname;
  if ( (p=getpwuid(s.st_uid)) == NULL) uidname = "?";
  else uidname = p->pw_name;
  char *gidname;
  struct group *g;
  if ( (g=getgrgid(s.st_gid)) == NULL) gidname = "?";
  else gidname = g->gr_name;
  printf("(%s,%s) %s\n",
         uidname,gidname,name);
}

int main(int argc,char *argv[]) {
  if (argc==1) ls(".");
  else while (--argc) ls(*++argv);
  exit(0);
}
```



```
$ ./lsid /usr/local/*
(yunes,admin) /usr/local/Cellar
(yunes,admin) /usr/local/Frameworks
(yunes,admin) /usr/local/Homebrew
(root,wheel) /usr/local/MacGPG2
(yunes,admin) /usr/local/bin
(yunes,admin) /usr/local/etc
(root,wheel) /usr/local/gfortran
(yunes,admin) /usr/local/gnat
(yunes,admin) /usr/local/go
(yunes,admin) /usr/local/include
(yunes,admin) /usr/local/lib
(yunes,admin) /usr/local/libexec
(root,wheel) /usr/local/man
(yunes,admin) /usr/local/opt
(yunes,admin) /usr/local/sbin
(yunes,admin) /usr/local/share
(yunes,admin) /usr/local/texlive
(yunes,admin) /usr/local/var
$
```

modification des horodatages : `utimes`

Definition (`utimes`)

```
#include <sys/time.h>
int utimes(const char *path, const struct
    timeval times[2]);
```

permet d'obtenir le positionnement des dates de dernier accès et modification du fichier concerné (sous réserve des droits nécessaires). Les dates sont exprimées relativement à `EPOCH`.
La fonction renvoie 0 en case de succès et -1 sinon (+`errno`).

`struct timeval`

Definition (`struct timeval`)

La structure doit comporter au moins les champs suivants :

`time_t tv_sec` un nombre de secondes;

`suseconds_t tv_usec` une précision en μ-secondes.

(quelques) fonctions sur les dates

Definition (`gettimeofday`)

```
#include <sys/time.h>
int gettimeofday(struct timeval *restrict tp
    , void *restrict tzp);
```

permet d'obtenir la date courante exprimée en durée depuis l'`EPOCH`. Le second paramètre doit être le pointeur nul. Cette fonction devrait disparaître dans le futur.

retour sur le C et les dates et temps

En C la manipulation des dates et temps reposent sur les définitions suivantes définies dans `<time.h>`. Types :

`clock_t` mesure de temps (type entier),

`time_t` mesure de temps (type entier),

`struct timespec` mesure de temps en nano-secondes,

`struct tm` temps du calendrier.

Constantes :

`CLOCK_PER_SEC` représente un nombre de tops d'horloge par seconde (une fréquence) de type `clock_t`;

`TIME_UTC` représente l'origine des mesures des temps UTC (type entier).

struct timespec

Definition (struct timespec)

Doit contenir au moins les champs suivants :

`time_t tv_sec` nombre de secondes

`long tv_nsec` nombre de nano-secondes

struct tm

Definition (struct tm)

Doit contenir au moins les champs suivants :

`int tm_sec` secondes

`int tm_min` minutes

`int tm_hour` heures

`int tm_mday` jour du mois (1 : premier jour)

`int tm_mon` mois (0 : janvier)

`int tm_year` année (0 : 1900)

`int tm_wday` jour de la semaine (0 : dimanche)

`int tm_yday` jour de l'année (0: 1er janvier)

`int tm_isdst` changement d'heure (booléen)

temps processeur utilisé `clock`

Definition (`clock`)

```
#include <time.h>
clock_t clock(void);
```

qui renvoie une approximation du temps processeur utilisé par le processus depuis le début de son exécution effective. En cas d'échec renvoie `(clock_t)-1`.

différence de temps `difftime`

Definition (`difftime`)

```
#include <time.h>
double difftime(time_t t1,time_t t0);
```

qui renvoie la différence entre $t1$ et $t0$

conversions d'une date de calendrier
mktime, gmtime, localtime

Definition (mktime)

```
#include <time.h>
time_t mktime(struct tm *ptr);
```

qui permet d'obtenir une mesure de temps à partir d'une date de calendrier.

La fonction «inverse»

Definition (gmtime)

```
#include <time.h>
struct tm *gmtime(const time_t *time);
```

qui permet d'obtenir une date de calendrier UTC à partir d'une mesure de temps.

La fonction «inverse»

Definition (localtime)

```
#include <time.h>
struct tm *localtime(const time_t *time);
```

qui permet d'obtenir une date de calendrier localisée (fuseau horaire) à partir d'une mesure de temps.

temps courant `time`, `timeget_spec`

Voir page 303 pour la fonction `time`.

Definition (`timespec_get`)

```
#include <time.h>
int timespec_get(struct timespec *ptr,int
    base);
```

qui permet d'obtenir la mesure de temps de la date courante. *base* doit être égal à `TIME_UTC`.

conversions avec des chaînes de caractères `ctime`, `asctime`, `strftime`

Definition (`ctime`)

```
#include <time.h>
char *ctime(const time_t *time);
```

qui permet d'obtenir une chaîne de caractères qui représente la date exprimée par la mesure (le fuseau horaire est pris en compte).
Cette fonction est équivalente à `asctime(localtime(time))`.

Definition (`asctime`)

```
#include <time.h>
char *asctime(const struct tm *date);
```

qui permet d'obtenir une chaîne de caractères qui représente la date exprimée par la date.

Definition (`strftime`)

```
#include <time.h>
size_t strftime(char *s,size_t size,const
   char *format,const struct tm *date);
```

qui permet d'obtenir une chaîne de caractères qui représente la date exprimée par la date selon un format choisi.
Consulter la documentation en ligne pour retrouver les spécifications de format possibles.

détermination des autorisations `access`

Definition (`access`)

```
#include <unistd.h>
int access(const char *path, int amode);
```

qui permet de déterminer si un accès décrit par *amode* est autorisé sur le fichier. Les valeurs pour *amode* sont une combinaison des valeurs :

- `R_OK` lecture
- `W_OK` écriture
- `X_OK` exécution
- `F_OK` test d'existence

Le fonctionnement de la fonction précédente repose sur le fait que tout processus est propriété d'un utilisateur et d'un groupe. Par conséquent, les droits d'accès sont déterminés par les droits de l'utilisateur si le propriétaire du fichier est le même que le propriétaire du processus, sinon par les droits du groupe si le propriétaire du processus est membre du groupe propriétaire du fichier ou que le groupe propriétaire du processus est le même que le groupe propriétaire du fichier et par les droits des autres sinon.