TD de Système n° 4

Exercice 1 : Masques de bits

Imaginons que vous êtes en train de coder un client mail. Spécifier une structure de données *compacte* `props_t` stockant les propriétés d'un mail :

- Lu/non-lu
- Important ou pas
- Spam ou pas
- Une parmi huit étiquettes (par exemple "travail", "personnel", "urgent", etc.)
- Une somme de contrôle du mail sur 2 bits

1. Écrire les fonctions d'accès à chaque propriétés comme des macros préprocesseur.
2. De la même façon, écrire les fonctions de modification de chaque propriétés.
3. Écrire une fonction

   ```
   props_t encoder(char lu, char important, char spam, char
   etiquette, char checksum);
   ```

   encodant toutes les propriétés vus comme des `char` dans votre structure de données.
4. Écrire les fonctions de lecture et d'enregistrement à partir d'un fichier :

   ```
   int lecture(FILE *f, props_t *p);
   int ecriture(FILE *f, props_t p);
   ```

Exercice 2 : « `chmod` »

1. Écrire la commande « `chmod` *fichier mode* » à l'aide de l'appel système du même nom :

   ```
   int chmod(const char *path, mode_t mode);
   ```

   en supposant que le *mode* est donné sous forme numérique (quadruplet de valeur octale). On rappelle que les trois premières valeurs octales sont le masque de :
   - 1 pour l'execution
   - 2 pour l'écriture
   - 4 pour la lecture
2. Modifier le programme pour accepter l'option « `--reference=`*fichier* » qui copie le mode depuis un autre fichier. On utilisera l'appel

   ```
   int stat(const char *path, struct stat *buf);
   ```
3. Modifier le programme pour accepter des modes sous forme symbolique (format « *a*`+`*b* » où *a* est `u`, `g` ou `o` et *b* est `r`, `w` ou `x`)

Exercice 3 : Arborescence fantôme

Le but de cet exercice est de copier un répertoire et récursivement tout ce qu'il contient, mais au lieu de créer des copies des fichiers sources, on créera des liens symboliques vers eux. On obtient ce qu'on appelle une «arborescence fantôme» puisqu'elle ne contient que son «squelette», pas sa «chair» (le contenu des fichiers). Cette technique est utilisée pour ne pas polluer l'arborescence source tout en ayant accès à ses fichiers, par exemple quand on compile un gros projet.

1. Ecrire une fonction récursive

   ```
   void parcours_dir(char *path);
   ```

   qui parcourt le répertoire `path` affiche le nom de son contenu et parcourt récursivement ses sous-répertoires. Vous aurez besoin des appels système

   ```
   DIR *opendir(const char *name);
   struct dirent *readdir(DIR *dirp);
   int stat(const char *path, struct stat *buf);
   int closedir(DIR *dirp);
   ```

2. Modifier la fonction précédente pour qu'elle crée une arborescence fantôme. Pour chaque membre du répertoire courant :
   – Si c'est un répertoire, elle créera un sous-répertoire dans l'arborescence fantôme,
   – Si c'est un fichier, elle créera un lien symbolique vers ce fichier.
   Vous aurez besoin de :

   ```
   int mkdir(const char *pathname, mode_t mode);
   int symlink(const char *oldpath, const char *newpath);
   ```

On rappelle les structures de données utiles définies dans la librairie standard :

```
struct dirent {
  ino_t         d_ino;      /* inode number */
  char          d_name[256]; /* filename */
};

struct stat {
  dev_t    st_dev;    /* ID of device containing file */
  ino_t    st_ino;    /* inode number */
  mode_t   st_mode;   /* protection */
  nlink_t  st_nlink;  /* number of hard links */
  uid_t    st_uid;    /* user ID of owner */
  gid_t    st_gid;    /* group ID of owner */
  dev_t    st_rdev;   /* device ID (if special file) */
  off_t    st_size;   /* total size, in bytes */
  blksize_t st_blksize; /* blocksize for file system I/O */
  blkcnt_t st_blocks; /* number of 512B blocks allocated */
  time_t   st_atime;  /* time of last access */
  time_t   st_mtime;  /* time of last modification */
  time_t   st_ctime;  /* time of last status change */
};
```