TD/TP de Système n° 7

On rappelle les structures de données utiles définies dans la bibliothèque 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 */
};
```

Ainsi que les fonctions :

```
DIR *opendir(const char *name);
struct dirent *readdir(DIR *dirp);
int stat(const char *path, struct stat *buf);
int closedir(DIR *dirp);
int chdir(const char *path);
int open(const char *path, int oflag, ...);
int close(int fildes);
size_t read(int fildes, void *buf, size_t size);
size_t write(int fildes, void *buf, size_t size);
off_t lseek(int fildes, off_t offset, int whence);
struct group *getgrgid(gid_t gid);
struct passwd *getpwuid (uid_t uid);
int chmod(const char *pathname, mode_t mode);
```

Exercice 1 : « pwd »

Le but de cet exercice est de réécrire la commande `pwd` qui affiche le chemin du répertoire courant.

Pour cela, on remontera dans l'arborescence jusqu'à la racine, en recherchant à chaque étape le numéro d'inœud du répertoire courant dans son répertoire-père pour déterminer son nom. Par souci de simplification, on supposera que l'arborescence n'est constituée que d'un seul disque logique (*device*), et que la racine de l'arborescence est donc le seul répertoire dont le numéro d'inœud est .

On suppose qu'on dispose d'une fonction `char *prepend(const char *s, char *t)` insérant une chaîne `s`

Exercice 4 : surveillance de fichier

En utilisant la date de modification, écrire un programme qui "surveille" un fichier fourni en argument jusqu'à ce que ce fichier soit modifié. Le programme devra consulter sans cesse la date de modification (avec `stat()`) jusqu'à ce qu'elle change ; puis quitter. On peut utiliser les fonctions `sleep()` ou `usleep()` entre deux appels à `stat()` afin que le programme ne consomme pas tout le temps CPU. Pour modifier un fichier, on peut faire `touch le_fichier`, ou encore `echo >> le_fichier` par exemple.

Tester le programme avec un répertoire. Quelles sont les opérations qui modifient un répertoire

Exercice 5 : « du »

La commande `du -k` utilisée avec un nom de fichier (au sens large) en argument affiche la taille en kilo octets occupée par l'arborescence A située sous ce fichier (sans suivre les liens symboliques) ainsi que les tailles des sous-arborescences des répertoires apparaissant dans A.

Programmer récursivement cette commande ; vous pourrez comparer le résultat de votre commande au résultat de `du -k` sur le répertoire `/ens/micheli/L3/Exdu`.

Testez maintenant votre commande sur le répertoire `/ens/micheli/L3/AutreExdu` et comparez au résultat de `du -k`. Pour comprendre où est le problème, vous pouvez utiliser l'option `-a` de la commande `du -k`, qui permet d'afficher la taille de chaque élément décompté dans la sous-arborescence. Comment faire pour le résoudre (On vous demande une réponse sans programmation ; vous pourrez essayer de l'implémenter à la fin de la séance si vous terminez le reste.

Exercice 6 : wc

Ecrire un programme `monwc` tel que `monwc -`*opt toto* affiche le nombre de :

- lignes du fichier *toto* si *opt* est l,
- mots du fichier *toto* si *opt* est u,
- caracteres du fichier *toto* si *opt* est m,
- octets du fichier *toto* si *opt* est c.