IUT Robert Schuman
Institut universitaire de technologie
Université de Strasbourg

S21 - Programmation système

TP n° 1

Filesystem

Objectifs : Comprendre l'utilisation des fonctions système permettant de naviguer dans le système de fichiers Unix.

Notions abordées : Programmation système en C

1 Commande ls

L'objectif de cet exercice est de récrire une fonction similaire à la commande système `ls`. La liste des fichiers contenus dans un répertoire peut être obtenue par l'appel à la fonction `scandir` (voir annexe et manuel en ligne). Cette fonction crée une liste de structures `dirent`, chacune représentant un des fichiers du répertoire passé en paramètre à `scandir`. À partir du nom de fichier contenu dans la structure `dirent`, il est possible d'obtenir plus d'informations sur le fichier (propriétaire, droits, type, etc.) en faisant appel à la fonction `stat` (voir annexe et manuel en ligne).

Liste de tous les fichiers

Coder la fonction de prototype :

```
int ls_a(const char* directory);
```

qui affiche la liste de tous les fichiers du répertoire `directory` dans l'ordre alphabétique. Cela doit correspondre à l'appel de la commande système `ls -a` (tous les fichiers, mêmes les fichiers cachés). La fonction renvoie `EXIT_FAILURE` en cas d'erreur et `EXIT_SUCCESS` sinon.

Attention : les noms de fichier sont séparés par 2 tabulations `"\t\t"` et il y a un retour à la ligne uniquement à la fin de la liste. Exemple d'exécution du programme de test :

```
# ./ls_a .
.       ..      file1.txt       ls.c            ls.h            makefile
```

Liste des fichiers non cachés

Coder la fonction de prototype :

```
char is_visible(const struct dirent* file);
```

qui renvoie `1` si le nom du fichier correspondant à la structure `dirent` passée en paramètre est un fichier non caché et `0` sinon. Utiliser cette fonction lors de l'appel à `scandir` (troisième paramètre) pour écrire la fonction de prototype :

```
int ls(const char* directory);
```

qui affiche la liste de tous les fichiers du répertoire `directory` dans l'ordre alphabétique. Cela doit correspondre à l'appel de la commande système `ls`. La fonction renvoie `EXIT_FAILURE` en cas d'erreur et `EXIT_SUCCESS` sinon.

```
# ./ls_ .
file1.txt       ls.c            ls.h            makefile
```

Récupération d'un nom d'utilisateur

Coder la fonction de prototype :

```
char* get_user(uid_t uid);
```

qui renvoie le nom de l'utilisateur dont l'`uid` est passé en paramètre. Si l'utilisateur n'existe pas, la fonction retourne `NULL`. La fonction `getpwuid` peut être utilisée pour cela.

Pour tester votre fonction, utiliser la commande `id` qui affiche l'identité de l'utilisateur courant avec notamment son `uid` et tous les groupes auxquels il appartient.

Récupération d'un nom de groupe

Coder la fonction de prototype :

```
char* get_grp(uid_t gid);
```

qui renvoie le groupe d'identifiant `gid` passé en paramètre. Si le groupe n'existe pas, la fonction retourne `NULL`. La fonction `getgrgid` peut être utilisée pour cela.

Récupération du type d'un fichier

Coder la fonction de prototype :

```
unsigned char get_type(unsigned long mode);
```

qui renvoie un caractère correspondant au type de fichier décrit par l'entier `mode` de la structure `stat`. La fonction retourne :

- 'B' pour un périphérique de bloc ;
- 'C' pour un périphérique de caractère ;
- 'D' pour un répertoire ;
- 'P' pour un tube ou une FIFO ;
- 'L' pour un lien symbolique ;
- 'F' pour un fichier ordinaire ;
- 'S' pour un socket ;
- '?' si le type n'est pas connu.

Récupération des droits sur un fichier

Coder la fonction de prototype :

```
void fmt_rights(unsigned long mode, char* rights);
```

qui remplit la chaîne de caractères `rights` (déjà allouée) à partir des informations sur les droits contenus dans l'entier `mode` de la structure `stat`. Les droits sont codés sur les 9 bits de poids faible de l'entier long `mode` (3 bits pour les droits en lecture/écriture/exécution du propriétaire, puis 3 bits pour ceux du groupe puis les 3 suivants pour les autres utilisateurs). On mettra un `'o'` lorsque le droit est actif et un `'-'` lorsqu'il ne l'est pas. Exemple : si les 9 derniers bits sont `0750` (en octal), la chaîne `rights` contiendra la valeur "oooo-o—".

Liste des fichiers avec leurs informations

Coder la fonction de prototype :

```
int ls_l(char* directory);
```

qui affiche la liste de tous les fichiers non cachés du répertoire `directory` par ordre alphabétique avec l'ensemble des informations liés à ces fichiers : type, droits, propriétaire, groupe, taille, date et heure de dernière modification, nom du fichier. La fonction renvoie `EXIT_FAILURE` en cas d'erreur et `EXIT_SUCCESS` sinon.

Pour être sûr de respecter l'affichage demandé, utiliser la fonction `print_line` donnée pour afficher chacune des lignes. Exemple d'affichage lors de l'appel de la fonction de test :

```
# ./ls_l .
F oooo-oo-o    wemmert      staff    50880 Wed Mar  2 13:20:04 2022 ls
F oo-o--o--    wemmert      staff     5480 Wed Mar  2 13:20:01 2022 ls.c
F oo-o--o--    wemmert      staff       42 Wed Jun 30 07:19:50 2021 ls.h
F oo-o--o--    wemmert   everyone       32 Wed Jun 30 07:19:04 2021 makefile
```

2 Commande pwd

Coder la fonction de prototype :

```
void pwd();
```

qui affiche le nom complet du répertoire courant. Exemple :

```
# ./pwd_
/adhome/w/we/wemmert/S21/tp
```

L'idée consiste à écrire une fonction intermédiaire :

```
int get_name_from_ino(char* dir, ino_t inode, char* dirname);
```

qui permet de retrouver le nom du fichier ayant le numéro d'inode `inode` dans le répertoire `dir`. Le résultat est copié dans la variable `dirname`.

3 Commande find

Coder la fonction de prototype :

```
int find(const char* dir, const char* str);
```

qui affiche la liste de tous les fichiers contenant la chaîne de caractères `str` dans leur nom et se trouvant dans l'ensemble de tous les sous-répertoires du répertoire passé en paramètre. La fonction renvoie `EXIT_FAILURE` si la chaîne `str` n'a pas été trouvée et `EXIT_SUCCESS` sinon.

4 Commande grep

Coder la fonction de prototype :

```
int grep(const char* dir, const char* str);
```

qui affiche l'ensemble des lignes des fichiers du répertoire `dir` qui contiennent la chaîne de caractères `str` passée en paramètre. La fonction renvoie `EXIT_FAILURE` si la chaîne `str` n'a pas été trouvée et `EXIT_SUCCESS` sinon.

Chaque ligne débute par le nom du fichier suivi de *':'* puis la ligne complète contenant la chaîne recherchée. Exemple pour l'appel de la commande `grep` avec la chaîne de caractères à trouver `"ls"` :

```
# ./grep_ . ls
ls.c:int ls_l(char* directory) {
ls.c:  else {
ls.c:    else
ls.c:    ls_l(argv[1]);
makefile:_ls: ls.c ls.h
makefile: gcc -o _ls ls.c
```

Fonctions et structures système à utiliser

```
#include <pwd.h>
#include <grp.h>
#include <dirent.h>

int scandir(const char * dir, struct dirent ***namelist,
    int (*filter)(const struct diren t *),
    int (*compar)(const struct dirent **, const struct dirent **));

struct dirent {
    ino_t          d_ino;       /* numéro d'inoeud */
```

```
    off_t          d_off;       /* décalage jusqu'à la dirent suivante */
    unsigned short d_reclen;    /* longueur de cet enregistrement */
    unsigned char  d_type;      /* type du fichier uniquement sur BSD pas Linux */
    char           d_name[256]; /* nom du fichier */
};
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *path, struct stat *buf);

struct stat {
    dev_t     st_dev;      /* ID du périphérique contenant le fichier */
    ino_t     st_ino;      /* Numéro inoeud */
    mode_t    st_mode;     /* Protection */
    nlink_t   st_nlink;    /* Nb liens matériels */
    uid_t     st_uid;      /* UID propriétaire */
    gid_t     st_gid;      /* GID propriétaire */
    dev_t     st_rdev;     /* ID périphérique (si fichier spécial) */
    off_t     st_size;     /* Taille totale en octets */
    blksize_t st_blksize;  /* Taille de bloc pour E/S */
    blkcnt_t  st_blocks;   /* Nombre de blocs alloués */
    time_t    st_atime;    /* Heure dernier accès */
    time_t    st_mtime;    /* Heure dernière modification */
    time_t    st_ctime;    /* Heure dernier changement état */
};
```