IUT Robert Schuman
Institut universitaire de technologie
Département **informatique**
Université de Strasbourg

S21 - Programmation filesystem

TP noté

Consignes

Ce TP noté est à réaliser en toute autonomie, sans aucun échange avec les autres étudiants, ni utilisation d'un navigateur internet. Vous pouvez utiliser vos codes développés durant les TPs faits durant le module et la documentation des fonctions standards via l'utilisation de la commande `man`. L'ensemble des fonctions utiles est rappelé en fin de sujet.

L'évaluation est faite par utilisation de la commande `push_code` vue lors des TPs. L'exercice s'appelle `s21_eval`.

```
./push_code s21_eval
```

Pensez à tout de suite tester la commande pour obtenir un premier rapport, avant même d'avoir commencé à coder.

L'évaluation est faite en comparant la sortie de votre programme avec la nôtre. N'ajoutez aucun `printf` non nécessaire ou appel à `perror` dans votre code.

1 Recherche d'une expression dans du code source

La première commande à réécrire est la commande Unix `ack`. Celle-ci recherche une expression dans tous les fichiers de l'arborescence depuis le répertoire courant (et tous ses descendants) et affiche les lignes la contenant. Ici, nous allons limiter la recherche aux fichiers sources en langage C, à savoir ceux ayant une extension `.c` ou `.h` et **uniquement ceux du répertoire courant sans descendre dans l'arborescence.**

Par exemple, si on souhaite rechercher l'utilisation de `printf` dans notre code, cela correspond à l'appel de la commande :

```
# ack --ignore-ack-defaults --type-add=cc:ext:c,h -t cc "printf"
main.c
5:    printf("\n--------------------------------------\n");

s21.c
26:            sprintf(full_path, "%s/%s", path, entry->d_name);
39:                printf("%llu\t%s\n", usage, full_path);
46:    printf("%llu\t%s\n", total_size, path);
60:    printf("%-28s %12lu %12lu %12lu %6s  %s\n", device, blocks, used, available,  per
70:    printf("%-28s %12s %12s %12s %6s  %s\n", "Filesystem", "1K-blocks", "Used", "Avai
91:       sprintf(percent, "%.0f%%", 100.0 * (1.0 - (float)buffer.f_bfree / (float)buffe
```

La commande affiche le nom du fichier uniquement si une instance de l'expression y a été trouvée puis toutes les lignes contenant l'expression en débutant par le numéro de la ligne. Les paramètres passés à la commande permettent simplement de ne pas chercher en profondeur dans l'arborescence et de chercher uniquement dans les fichiers `.c` et `.h`.

Commencer par coder la fonction de prototype :

```
uint8_t is_c(char* file_name);
```

qui renvoie `1` si le fichier dont le nom `file_name` est passé en paramètre est un fichier source en langage C (son extension est soit `.c`, soit `.h`). On considérera que `file_name` est une chaîne de caractères bien formée et on ne testera pas l'existence ou les droits d'ouverture du fichier. Le test se base uniquement sur le nom du fichier et non son contenu.

Ensuite, coder la fonction de prototype :

```
void print_lines(char* file_name, char* expr);
```

qui affiche le nom du fichier puis l'ensemble des lignes du fichier `file_name` contenant l'expression `expr` avec son numéro en en-tête de ligne. Exemple pour le fichier `s21.c` et l'expression `"printf"` :

```
s21.c
26:             sprintf(full_path, "%s/%s", path, entry->d_name);
39:                 printf("%llu\t%s\n", usage, full_path);
46:     printf("%llu\t%s\n", total_size, path);
60:     printf("%-28s %12lu %12lu %12lu %6s  %s\n", device, blocks, used, available,  per
70:     printf("%-28s %12s %12s %12s %6s  %s\n", "Filesystem", "1K-blocks", "Used", "Avai
91:        sprintf(percent, "%.0f%%", 100.0 * (1.0 - (float)buffer.f_bfree / (float)buffe
```

Attention : La fonction ne teste pas s'il s'agit d'un fichier `.c` ou `.h`. Si le fichier ne contient pas l'expression recherchée, rien ne s'affiche, pas même le nom du fichier. Idem si le fichier n'existe pas ou ne peut pas être lu.

Finalement, coder la fonction de prototype :

```
void ack(char* expr);
```

qui affiche pour chaque fichier de l'arborescence depuis le répertoire local contenant l'expression `expr`, l'ensemble des lignes la contenant, précédées par leur numéro respectif. Voir l'exemple de la commande `ack` ci-dessus. Votre commande doit donner exactement le même résultat. Si l'expression n'est pas trouvée, rien ne s'affiche.

On ne tiendra pas compte de l'affichage en couleur de la commande Unix. Le résultat est à afficher simplement en noir et blanc. Il y a un saut de ligne pour chaque nouveau fichier (ligne vide) et un saut de ligne en fin de commande.

Le format pour l'affichage d'une ligne est `printf("%d: %s", le_numéro_de_ligne, la_ligne);`

2 Espace utilisé par un fichier ou un répertoire

Pour connaître l'espace disque utilisé par un fichier ou un répertoire, il existe la commande système `du`.

Attention, l'**espace utilisé par un fichier** ne correspond pas à sa taille, mais au nombre de blocs qu'il occupe multiplié par la taille d'un bloc.

Un bloc système fait une taille de `4096` octets, mais le champ `st_blocks` indique le nombre de blocs de `512` octets alloués au fichier (cette valeur peut être inférieure à `st_size/512` si le fichier contient des trous).

Lors de l'appel à `du`, l'arborescence complète du répertoire est parcourue en profondeur pour en calculer **son poids total en Kio**. Le poids correspond à la somme de l'espace utilisé par chaque fichier. Tous les fichiers sont pris en compte, même les fichiers cachés et **même les répertoires** qui sont aussi des fichiers en Unix et occupe de l'espace.

Voici un exemple de ce qu'on obtient avec le répertoire `test_du` fourni sur la machine `troglo.iutrs.unistra.fr` (on ajoute l'option `-a` à la commande `du` pour obtenir le comportement demandé) :

```
# du -a test_du
108  test_du/img/stp 2.png
1236 test_du/img/unistra.eps
8    test_du/img/router.png
32   test_du/img/plan_adressage.png
8    test_du/img/flash.png
48   test_du/img/stp.png
260  test_du/img/computer_gray.png
112  test_du/img/plan_adressage_correction.xcf
4    test_du/img/laptop.eps
224  test_du/img/configIP.svg
56   test_du/img/archi_TP.png
140  test_du/img/plan adressage2.png
2240 test_du/img
12   test_du/.DS_Store
120  test_du/2022 - tp noté.pdf
8    test_du/s21-start.zip
2384 test_du/
```

Si le nom passé en paramètre correspond à un fichier et non un répertoire, une seule ligne s'affiche avec la taille du fichier :

```
# du main.c
4       main.c
```

Coder la fonction de prototype :

```
unsigned long du_file(char *file_name);
```

qui affiche l'équivalent du résultat de l'appel à la commande `du` avec en paramètre le nom d'un fichier `file_name`. La fonction renvoie la taille en Kio utilisée pour le stockage de ce fichier. Si le fichier n'existe pas, rien ne s'affiche. On fera l'hypothèse que le nom passé en paramètre correspond à un fichier, aucun test n'est à faire pour le vérifier.

Coder la fonction de prototype :

```
unsigned long du(char *path);
```

qui affiche l'équivalent du résultat de l'appel à la commande `du` avec en paramètre un nom de fichier ou répertoire `path`.

L'**ordre d'affichage des fichiers** n'est pas contrôlé par la commande `du`. Nous utiliserons l'ordre alphabétique pour le parcours des fichiers des répertoires pour s'assurer d'un ordre toujours similaire. Pour cela, utiliser le paramètre `alphasort` lors de l'appel à la fonction `scandir`.

Le format d'affichage d'une ligne est `printf("%lu\t%s\n", la_taille, le_répertoire_ou_fichier);`

Fonctions et structures à utiliser

Consultez le `man` pour plus d'informations sur ces fonctions.

Liste des fichiers d'un répertoire

```c
#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 */
};
```

Informations sur un fichier

```c
#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 */
};
```

Lecture du contenu d'un fichier

```c
#include <stdio.h>

/* ouverture d'un fichier */
FILE * fopen(const char * restrict path, const char * restrict mode);

/* fermeture d'un fichier */
int fclose(FILE *stream);

/* lecture d'une ligne d'un fichier ouvert */
char * fgets(char * restrict str, int size, FILE * restrict stream);

/* retour au début d'un fichier ouvert */
void rewind(FILE *stream);
```

```c
/* déplacement à une certaine position dans un fichier */
int fseek(FILE *stream, long offset, int whence);
```

Manipulations des chaînes de caractères

```c
#include <string.h>

/* La fonction strstr() cherche la première occurrence de la
  sous-chaîne aiguille au sein de la chaîne meule_de_foin.
  Elle renvoie un pointeur sur le début de la sous-chaîne,
  ou NULL si celle-ci n'est pas trouvée */
char *strstr(const char *meule_de_foin, const char *aiguille);

/* construction d'une chaîne à partir d'un format d'affichage */
int sprintf(char * restrict str, const char * restrict format, ...);

/* concanténation de deux chaînes */
char * strcat(char *restrict s1, const char *restrict s2);
```