Université Paris 7
Licence 3 Informatique – Systèmes
11 janvier 2012
Durée : 3 heures. Livres et photocopies de livres interdits. Ordinateurs et téléphones portables interdits.
Les polys de cours et une feuille de notes personnelles manuscrite autorisés.

Le sujet comporte 6 pages.

Votre code doit être écrit de façon lisible, avec des indentations et des accolades appropriées permettant de voir la fin des blocs de code (fin de boucles, etc.).

Comprendre les programmes.

Exercice 1

Que contient le fichier `toto` après l'exécution du programme suivant :

```
1  int main(int argc, char *argv[]){
2    int d;
3    char tab[]="alamakota";
4    d = open("toto", O_WRONLY|O_TRUNC|O_CREAT|O_APPEND, S_IRUSR|S_IWUSR);
5    if(d == -1){
6      perror("open");
7      exit(EXIT_FAILURE);
8    }
9    write(d, &tab[5],4);
10   write(STDOUT_FILENO, &tab, 3);
11   dup2(d,STDOUT_FILENO);
12   write(STDOUT_FILENO, &tab[3], 2);
13   write(d,&tab,5);
14   close(d);
15   exit(EXIT_SUCCESS);
16 }
```

Qu'est-ce que ce programme affiche sur le terminal ?

Exercice 2

Quel est le contenu du fichier `gaga` après l'exécution du programme suivant :

```
1  int main(void){
2    int desc;
3    char *tab="abcdefghij";
4    char buf[100];
5    desc = open("gaga", O_APPEND|O_CREAT|O_RDWR|O_TRUNC, S_IRUSR|S_IWUSR);
6    if( desc <  0 ){
```

```
7       perror("open");
8       exit(1);
9     }
10    write(desc,tab,strlen(tab));
11    lseek(desc, (off_t) 0, SEEK_SET);
12    read(desc,buf,4);
13    write(desc,buf,4);
14    exit(0);
15  }
```

Supposons que la ligne 5 du programme est remplacée par

```
desc = open("gaga", O_CREAT|O_RDWR|O_TRUNC,S_IRUSR|S_IWUSR);
```

Quel sera le contenu du fichier gaga si on exécute le programme ainsi modifié ?

Exercice 3

On considère le fichier `toto` qui se trouve dans le répertoire `/home/dupont`. En utlisant `ls -l` on trouve les propriétaires et les droits d'accès suivants pour `home`, `dupont` et `toto` :

```
drwxr-xr-x   3 root root  4096 2010-09-29 20:02 home
drwx--x--x 54 dupont ens 4096 2012-01-10 09:44 dupont
-rw-------  1 dupont ens   12 2012-01-10 11:28 toto
```

On suppose que l'utilisateur `jean` est différent de `root` et de `dupont` (et qu'il n'appartient pas au groupe `root`).

Question 1: L'utlisateur `jean` exécute la commande

```
cat /home/dupont/toto
```

Est-ce que cette tentative de lecture de `toto` lui permet d'afficher le contenu de ce fichier ?

Question 2: Maintenant `jean` exécute

```
ls /home/dupont
```

Quel résultat cela donne-t-il ?

Question 3: Et finalement il exécute

```
rm /home/dupont/toto
```

Est-ce qu'il réussit de supprimer `toto` ?

Écrire des fonctions.

```
Dans toutes les programmes à écrire vous pouvez utiliser toutes les
fonctions POSIX qui conviennent à l'exception de la fonction
                        system
dont l'utilisation est prohibée.

Il est inutile d'écrire les  include.
```

Exercice 4

Le but de cet exercice est d'écrire une version simplifiée de la commande `mv` qui déplace les fichiers (aussi bien ordinaires que spéciaux) d'un répertoire à l'autre.

L'écriture de la commande sera divisée en plusieurs questions, chaque question demande d'écrire une fonction particulière. Il n'est pas nécessaire de faire les questions dans l'ordre. Si vous écrivez une fonction en réponse à la question i vous pouvez toujours utiliser les fonctions demandées dans les questions précédentes même si vous ne les avez pas encore écrites. Donc inutile de s'attarder sur une question qui vous pose problème, passez plutôt à la question suivante et revenez plus tard si le temps le permet.

Remarques. Dans vos programmes vous pouvez utiliser librement la fonction

```
char *concatenation(char *s,...)
```

(on suppose qu'elle est donnée, on ne demande pas de l'écrire). Cette fonction, avec un nombre variable d'arguments, concatène les chaînes de caractères passées en argument. Par exemple

```
concatenantion("/home/dupont/bin", "/", "toto", (char *)NULL)
```

retournera une chaîne de caractères `"/home/dupont/bin/toto"` Notez que le dernier paramètre de `concatenation` doit être toujours `NULL` (nécessaire pour déterminer le nombre d'arguments).

```
concatenation("toto",(char *)NULL)
```

retournera juste une copie de la chaîne `"toto"`. On suppose que la fonction `concatenation` retourne l'adresse vers une mémoire statique que les appels suivants peuvent modifier.

D'autres fonctions qui peuvent être utiles :

```
char *dirname(char *chemin);
char *basename(char *chemin);
```

`basename` retourne le dernier élément d'un chemin bien formé (chemin bien formé signifie des noms séparés par /), la fonction `dirname` retourne le chemin après la suppression de basename. Les deux fonctions peuvent modifier le chemin passé en argument, donc avant de les appliquer il convient de copier le chemin dans un tampon pour appliquer les fonctions sur ce tampon. Par exemple pour les trois chemins

```
char *s1="/home/dupont/toto";
char *s2="/home/toto/";
char *s3="toto";
```

`basename` retourne dans les trois cas `"toto"` et `dirname` retourne respectivement `"/home/dupont"`, `"/home"` et `"."` (une chaîne avec un point). Donc en concaténant le résultat de `dirname` appliqué à un chemin, la chaîne `"/"` et le résultat de `basename` appliqué au même chemin, on retrouvera un chemin équivalent au chemin initial.

Dans la suite, à chaque fois que l'on parle de fichier, cela veut dire fichier au sens large : fichier régulier, répertoire, lien symbolique, tube nommé sont tous des fichiers. Si on veut parler d'un type particulier de fichier on précisera toujours.

Question 1: Écrire la fonction

```
int mv_file(char *source, char *rep)
```

Cette fonction déplace le fichier régulier `source` dans le répertoire `rep`. La fonction doit procéder de la façon suivante :

- la fonction copie le fichier régulier `source` vers le répertoire `rep` en gardant le nom du fichier,
- ensuite la fonction supprime le fichier régulier `source`,
- la fonction retourne 0 si l'opération a réussi et -1 sinon.

Par exemple `mv_file("/home/toto/prog.c", ".")` déplace le fichier `prog.c` du répertoire `/home/toto` vers le répertoire courant. Le fichier créé dans le répertoire courant portera le même nom `prog.c`. On supposera que les paramètres de `mv_file` sont corrects (on ne demande pas de vérifier si `source` est un fichier régulier et `rep` un répertoire).

Question 2: Écrire la fonction

```
int mv_symlin(char *source, char *rep)
```

Cette fonction déplace le lien symbolique `source` vers le répertoire `rep`. La fonction procède de la façon suivante :

- on lit le contenu du lien symbolique `source` et on crée dans `rep` un lien symbolique avec le même nom et le même contenu que `source`,
- on supprime `source` (c'est le même appel système qui supprime les fichiers réguliers),
- on retourne 0 en cas de succès et -1 si le déplacement échoue.

Donc `mv_symlin("/home/toto/coto", "./bin")` crée dans le répertoire `./bin` un lien symbolique dont le nom est `coto` et dont le contenu est le même que le contenu du lien `/home/toto/coto`. Comme dans la question précédente, on assume que les paramètres de `mv_symlinl` sont corrects et on ne demande pas de les vérifier.

Question 3: Écrire la fonction

```
int mv_fifo(cher *source, char *rep)
```

Le paramètre `source` est un chemin vers un tube nommé. La fonction crée dans le répertoire `rep` un tube nommé de même nom que le tube référencé par `source` et supprime le tube `source` (pour la suppression de tubes nommés on utilise la même fonction qui est utilisée pour supprimer les fichiers réguliers).

Question 4: Écrire la fonction

```
int type_fichier(char *chemin)
```

qui retourne

(a) 0 si `chemin` est un répertoire,

(b) 1 si `chemin` est un fichier régulier,

(c) 2 si `chemin` est un lien symbolique,

(d) 3 si `chemin` est un tube nommé,

(e) -1 si aucun des cas précédents n'est valide.

Question 5: Écrire la fonction

```
int meme(char *premier, char *deuxieme)
```

Cette fonction retournera

(a) 2 si `premier` et `deuxieme` sont des chemins vers le même fichier,

(b) 1 si `premier` et `deuxieme` sont des chemins vers des fichiers différents mais les deux fichiers résident dans le même système de fichiers (c'est-à-dire le même volume logique, ou la même partition),

(c) 0 si les deux fichiers résident dans des systèmes de fichiers différents,

(d) -1 si on ne peut pas vérifier où réside un de ces fichiers (suite à un appel à `lstat` qui a échoué).

Rappel. Le même fichier veut dire le même numéro de i-node et le même volume logique (même système de fichier), par contre `cg` et `cd` peuvent être complètement différents en tant que chaînes de caractères.

Question 6:

Préambule. Les fonctions que vous avez eu à écrire dans les questions 1,2,3 sont inutilement compliquées si nous voulons déplacer un fichier simple [1]

entre deux répertoires qui sont dans le même système de fichiers (dans la même partition logique). Par exemple si nous voulons déplacer un fichier simple `/home/toto/ff` vers le répertoire `./bin` et que les deux répertoires `/home/toto` et `./bin` résident dans la même partition (voir la fonction précédente), alors il suffit

(a) de créer dans `./bin` un lien dur dont le nom est `ff` et qui référence le même fichier que `/home/toto/ff`,

(b) et ensuite de supprimer `/home/toto/ff`.

Donc tout se résume à la création d'une nouvelle entrée dans le répertoire `./bin` et la suppression d'une entrée du répertoire `/home/toto`, i.e. un simple appel à `link` suivi d'un appel à `unlink`, pas besoin de créer une copie physique du fichier.

Fin de préambule.

Écrire la fonction

```
int mv_tout(char *source, char *rep)
```

qui déplace `source` vers `rep` en implémentant l'algorithme suivant :

(1) si `rep` n'est pas un répertoire alors la fonction retourne immédiatement -1,

(2) si `source` est un fichier simple (fichier régulier, lien symbolique ou tube nommé) et que le répertoire « `dirname` de source » et le répertoire « `rep` » résident les tous les deux dans la même partition (pour le vérifier on utilisera la fonction de la question précédente), alors on déplace `source` vers `rep` en utilisant la méthode décrite dans le préambule et la fonction retourne 1,

(3) si `source` est un fichier simple (fichier régulier, lien symbolique ou un tube nommé) et que les répertoires « `dirname` de `source` » et « `rep` » résident dans des partitions différentes, alors on déplace `source` vers `rep` en utilisant une des trois fonctions des questions 1,2,3, selon le cas,

1. Ici et par la suite un « fichier simple » signifie soit un fichier régulier, soit un lien symbolique, soit un tube nommé. Un répertoire n'est pas un fichier simple.

(4) si `source` est un répertoire, alors on déplace récursivement tous les fichiers et sous-répertoires de `source` vers `rep` en utilisant l'algorithme suivant :

on parcourt le répertoire `source` et pour chaque entrée `XXX` de ce répertoire différente de `.` et de `..` :

(a) si `source/XXX` est un fichier simple, alors on fait un appel récursif à `mv_tout(source/XXX, rep)` (`source/XXX` désigne la concaténation des chaînes « `source` », `"/"` et « `XXX` »),

(b) si `source/XXX` est un répertoire, alors
- on crée un sous-répertoire `XXX` dans `rep`,
- ensuite on fait un appel récursif à `mv_tout(source/XXX, rep/XXX)`,
- et finalement on supprime le répertoire `rep/XXX` .

Dans les cas (2),(3),(4) la fonction `mv_tout` retourne 1.