Système d'Exploitation Unix
TP
Les appels système : gestion de fichiers

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction `main (int argc, char *argv [])`. Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affiches avec `perror`. Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`

Question 1 Création de fichiers `open ()`

a) Ecrire un programme qui crée un fichier en lecture/écriture au travers de l'appel

```
int open(const char *pathname, int flags, mode_t mode);
```

Si le fichier existe déjà, une erreur doit être retournée.

b) Quel est le code d'erreur retourné lorsque le fichier existe déjà ?

Si l'argument `mode` spécifié est `755` (`rwxr-xr-x`), est-ce que le fichier est créé avec exactement ces droits ? Expliquer !

Question 2 Caractéristique d'un fichier `stat(), fstat(), lstat`

Ecrire un programme qui récupère les caractéristiques de fichiers donnés, au travers des appels des fonctions `int stat(); int fstat();` et `int lstat;`. Pour un fichier donné, afficher les caractéristiques suivantes :

- le numéro d'inode,
- la protection,
- le nombre de liens physiques,
- l'ID du propriétaire,
- l'ID du groupe,
- la taille du fichier,
- la taille de bloc,
- le nombre de blocs,
- l'heure du dernier accès.

L'affichage d'une heure dans un format lisible peut être accompli en utilisant la fonction `ctime()`.

Question 3 Utilisation d'un fichier `open(), read(), write()`

a) Ecrire un programme qui recopie un fichier source, `fichier_source`, dans un fichier destinataire, `fichier_destinataire`. Le programme doit vérifier que le fichier source est un fichier régulier (utiliser la macro `S_ISREG(m)`, cf. `int stat()`). Le programme doit également vérifier qu'il n'existe pas déjà de fichier de même nom que le fichier destinataire.

b) Quels sont les temps d'exécution (utiliser `/bin/time`) respectifs si la taille du buffer utilisé dans la fonction `read()` est de 1024 octets puis un octet ? Expliquer !

Question 4 Duplication des descripteurs de fichier `dup(), dup2()`

a) Ecrire un programme qui redirige la sortie d'erreur standard vers un fichier, `fichier_erreur`, préalablement créé. C'est à dire, toute écriture de la forme `write(2, …)` doit se faire dans le fichier `fichier_erreur`. Le descripteur de valeur 2 étant au départ celui de la sortie d'erreur standard.

b) Quelle est la propriété de la fonction `dup()` qui est exploitée pour ainsi rediriger les E/S standards ?

c) Modifier le code du mini shell afin qu'il prenne en charge la redirection des entrées (**<**) et des sorties (**>**).

Question 5 Verrouillage des fichiers `fcntl()`

Ecrire un programme `lock_file` qui illustre la mise en oeuvre et le fonctionnement des verrous sur les fichiers en utilisant la primitive fcntl.
Lancement du programme:

```
./lock_file nom_fichier type_verrou debut_zone longueur_zone
```

Exemple:
- pour un verrou partagé

```
./lock_file test s 10 5
```

- pour un verrou exclusif

```
./lock_file test x 10 5
```

NB: le fichier test doit exister

Question 6 Déplacement de la tête de lecture/écriture des fichiers `lseek()`

a) Ecrire un programme qui crée un fichier vide. Positionner la tête de lecture/écriture sur le 10 000ème octet à partir du début du fichier. Ecrire un caractère à cette position.

b) Quelle doit être la taille du fichier ? Est-ce cette taille qui est retournée par la commande `ls -l` ? Est-ce que les blocs correspondant au trou de 9 999 caractères ont été alloués (utiliser df) ?
La primitive `lseek()` permet de déplacer l'offset courant d'un fichier et retourne le nouvel `offset`.

c) Ecrire un programme qui, après un certain nombre de lecture/écriture sur un fichier, retourne la valeur de l'`offset` courant.
L'`offset` est associé à un fichier et non pas à un descripteur. Si deux descripteurs référencent un même fichier, la modification (par lecture/écriture ou `lseek()`) de l'offset du fichier via un descripteur est "visible" via l'autre descripteur.

6.d) Vérifier l'affirmation précédente.

Question 7 Manipulation de répertoires `opendir(),readdir(), mkdir()`

La primitive `DIR *opendir(const char *pathname)` permet d'ouvrir en lecture le répertoire référencé par pathname.

La primitive `struct dirent *readdir(DIR *dp)` permet de lire l'entrée suivante du répertoire identifié par `dp`.

a) Ecrire un programme qui ouvre un répertoire (/tmp par exemple) et qui affiche tous les fichiers qui y sont contenus ainsi que le type de chacun (champ d_type de la strucure dirent).

La primitive `int mkdir(const char *pathname, mode_t mode)` permet de créer le répertoire de nom référencé par pathname.

b) Modifier le programme précédent en y ajoutant à la fin la création (dans le répertoire /tmp) d'un répertoire ayant pour nom votre propre nom.

c) Vérifier que le répertoire a bien été créé.

La primitive `void rewinddir(DIR *dp)` permet de positionner le pointeur de lecture dans un répertoire sur la première entrée de ce répertoire.

a) Modifier le programme précédent en y ajoutant à la fin 1) le positionnement en début du pointeur de lecture du répertoire et 2) l'affichage à nouveau du contenu du répertoire (/tmp).

Avant chaque exécution, supprimer du répertoire /tmp le répertoire que vous avez créé.

b) Modifier le programme précédent en y ajoutant à la fin la suppression du répertoire créé.

c) Vérifier que le répertoire a bien été supprimé.