Programmation Système, TD 2

Préambule

Rappel : Comme indiqué au TD 1, un nouveau répertoire est à créer pour ce TD, répertoire au sein duquel sera créé pour chaque exercice un nouveau répertoire associé nommé `Exo_xx` où `xx` désignera le numéro de l'exercice. A l'intérieur de chacun de ces répertoires, deux nouveaux répertoires seront créés :

— l'un, appelé `Include`, sera destiné à recevoir scripts, en-têtes, fichiers de paramètrage

— l'autre, appelé `WorkDir`, sera le répertoire de travail contenant codes sources et `makefile`

Afin de ne pas occuper inutilement de la place dans vos répertoires, lorsque vous finirez ce TD

1. penser à effacer tous les exécutables et `.o`
   — soit *via* la commande `make clean` accessible à partir du Makefile de votre répertoire de travail ;
   — soit à partir de la simple commande `rm */*/*.{o,a,exec}`, toujours dans votre répertoire de travail ; cette procédure sera répétée à chaque TD.
2. compresser le contenu de chaque répertoire associé à un exercice sous la forme d'une archive `.tar.gzip` *via* (par exemple) la commande

   ```
   tar cvf - exo_01 | gzip -9 - > exo_01.tar.gz
   ```

   pour l'exercice 01.

Des questions vous sont posées tout au long du TP : Les réponses sont à rendre sur une feuille de papier libre comportant votre nom et votre groupe.

Recopie d'un fichier

La recopie d'un fichier fait appel à l'utilisation des fonctions `open()`, `read()`, `write()`, et `close()`

Exercice 1. On souhaite proposer une primitive de recopie de fichier (qui prendra ici le nom de l'exo), de format

```
exo_01.exec {mode}{source}{destination}
```

dans lequel `mode` est pris dans les valeurs possibles `{char, block, all}` qui permettent de préciser comment effectuer la copie, `source` est le nom du fichier à copier et `destination` le nom de la copie. Pour réaliser l'écriture de cette primitive :

— Récupérer le répertoire compressé `CanevasTP_Fichiers.tar.gz` et le décompresser. Ce répertoire sert de base pour l'ensemble des exercices. Renommer `ModeleExo.cxx` en `exo_01.cxx`, étudier `nsSysteme.h` et `nsSysteme.cxx`. *Question 1 : Donner la liste des wrappers qui sont introduits.*

— Ecrire le wrapper (déclaration et définition) de la fonction `write()` dans l'espace de noms `nsSysteme` du fichier `nsSysteme.h` (consulter la section 2 du man *via* la commande `man 2 write`)

— Dans un nouvel espace de noms `nsFctShell` du fichier `nsSysteme.cxx`, définir la fonction `FileCopy(const char *dest, const char *source, const size_t NbBytes);` qui

1. ouvre le fichier source en mode lecture à l'aide d'un appel système
2. ouvre le fichier destination en mode écriture, création et troncature l'aide d'un appel système

3. réserve un tampon mémoire de `NbBytes+1` octets
4. dans une boucle, tant qu'on peut lire à partir du fichier source (i.e. l'appel système `Read()` n'est pas nul) lit (au plus) `NbBytes` octets dans le tampon (consulter `man 2 read`)
5. écrit le nombre d'octets ainsi lus, depuis le tampon, dans le fichier destination avec `Write()`
6. ferme enfin les deux fichiers l'aide des appels système adéquats

— Dans le fichier `exo_01.cxx`, définissez dans l'espace de noms anonyme la fonction de profil

```
size_t GetSizeInBytes(const string &How, const string &FileName)
```

qui prend en arguments le nom d'un fichier (`FileName`) et la manière (`How`) dont on veut effectuer la recopie d'un fichier, et retourne la taille des morceaux du fichier en octets, associée à la manière choisie. Plus précisément, on propose de pouvoir effectuer la recopie, au choix, caractère par caractère, bloc par bloc du support, ou enfin d'un seul tenant. Dans le premier cas, la valeur passée à l'argument `How` (à partir des arguments passés au programme par l'utilisateur) est `"char"`, dans le second cas `"block"`, dans le troisième cas `"all"`. Pour pouvoir déterminer la taille en nombre d'octets à renvoyer, `GetSizeInBytes()` utilise un appel à `Stat()` afin de récupérer les informations correspondants à partir des champs adéquats de la struct `struct stat` associée (voir le cours *Fichiers* et `man 2 stat`). En particulier, la taille d'un bloc est extraite du champs `.st_blksize` de la struct, alors que taille du fichier est obtenue à partir du champs `.st.size` de la même struct.

— Toujours dans le fichier `exo_01.cxx`, et une fois l'étape précédente terminée, passer à l'écriture de la fonction `main()` qui, dans l'ordre

1. prend (donc) trois arguments (comment recopier à partir de `"char"`, `"block"`, ou `"all"`, le nom du fichier source, et le nom du fichier destination
2. appelle `GetSizeInBytes()` pour connaitre la taille en octets du tampon à utiliser pour copier la source
3. affiche la taille du tampon ainsi obtenue
4. appelle enfin `FileCopy()` pour copier le fichier source sur la destination en utilisant la taille du tampon retournée par `GetSizeInBytes()`

— Compiler le programme, et l'essayer pour chacune des valeurs du premier paramètre, sur un fichier assez grand, par exemple l'exécutable du shell bash (vérifier avec la commande `which bash` que le chemin associé est `/usr/bin/bash`)

— Afin de mesurer les temps d'exécution associés aux différents paramètres qui sont passés au programme utiliser la commande `time`, par exemple avec le paramètre `all` et sur l'exécutable `bash` : `time ./exo_01.exec all /usr/bin/bash/ CopieBash`

Question 2 : Quels sont les temps d'exécution pour la recopie de l'exécutable `bash` *avec toutes les options ?*

Exercice 2. Drapeaux supplémentaires associés à l'appel d'`Open()` :

— Ajouter *temporairement* le drapeau supplémentaire `O_SYNC` dans l'appel d'`Open()` qui ouvre le fichier destination. L'option `O_SYNC` force l'écriture physique de la destination à chaque appel de `Write()`. Le temps supplémentaire peut devenir écrasant pour un mauvais choix de taille de tampon ! Compiler et tester avec la commande `time` pour constater la différence. *Question 3 : Quels sont dans ce cas les temps d'exécution pour la recopie de l'exécutable* `bash` *avec toutes les options ?* Ne pas oublier d'effacer l'option `O_SYNC`

— Ajouter temporairement le drapeau `O_APPEND` dans l'appel d'`open()` qui ouvre le fichier destination. Ecrire un fichier source `test.txt` constitué du seul text `"test open"`. Compiler et tester alors deux fois sur le fichier `test.txt` et vérifier le contenu de la destination pour observer la différence. Ne pas oublier d'effacer le drapeau. *Question 4 : Quelle est la différence avec ou sans l'emploi de l'option* `O_APPEND`

Effacement d'un fichier

Le système étant multi-utilisateur et multi-processus, les fichiers peuvent être manipulés de manière concurrente. Cette concurrence impacte le fonctionnement de la primitive `unlink()` permettant l'effacement d'un fichier.

Exercice 3. Appliquer la succession d'événements suivante sur un fichier :

1. ouvrir deux xterms, créer un fichier de plusieurs lignes (par exemple en tapant dans le shell `echo Coucou >> ~/Test.txt` quelques dizaines de fois – touche *flêche vers le haut* pour rappeler la dernière commande)
2. dans un des xterms, taper `less ~/Test.txt` et laisser l'affichage en suspend
3. dans l'autre xterm, lister le fichier à partir de la commande `ls -l`, puis l'effacer avec `rm` et essayer de le lister à nouveau
4. revenir au premier xterm, y appliquer la commande `less`. *Question 5 : Que constatez-vous et comment l'expliquez-vous ?*

Exercice 4. On étudie le mécanisme système qui permet l'implémentation de cette concurrence. Après écriture du wrapper `Unlink()`, cet exercice doit permettre l'effacement d'un fichier avant de procéder à des tests d'accès après destruction.

— Après avoir dupliqué le répertoire `Exo_01`...
— Dans le fichier `nsSysteme.h`, rajouter la déclaration et la définition de `Unlink()` (wrapper de `unlink()`).
— Dans le fichier `exo_05.cxx`, écrire la fonction `main()` qui prend comme argument un fichier de caractères et vérifie la présence de ce fichier dans le répertoire courant *via* un appel à `Stat()`.
— Ouvre le fichier en question.
— Efface ce fichier avec `Unlink()`
— Vérifie à présent l'absence de ce fichier dans le répertoire courant *via* un nouvel appel à `Stat()`.
— Ecrit sur la sortie standard (de descripteur de fichier 1) les dix premiers charactères du fichier effacé.

Question 6 : A partir des informations fournies par le man, expliquer pourquoi en dépit d'un appel à `Unlink()`*, l'affichage reste possible. Grâce à quelle variable réussi-t'on à copier le fichier ?*