Série d'exercices 7 : Fichiers et stockage

7.1 Sous Linux, l'utilitaire « file » retrouve le format d'un fichier. Comment fonctionne cet utilitaire? Que se passe-t-il en changeant l'extension du fichier?

L'utilitaire « file » lit le début du fichier et le compare avec une liste d'entêtes connus. Contrairement à l'explorateur Windows qui se base sur l'extension du fichier, l'utilitaire « file » retourne le type du contenu du fichier, peu importe son extension.

7.2 Dans un shell Linux, la variable d'environnement $PWD contient le chemin d'accès du répertoire courant. Utilisez cette variable pour construire un chemin absolu pour accéder au fichier « foo » se situant dans le répertoire parent.

« $PWD/../foo » : La variable $PWD contient le chemin absolu du répertoire courant. À partir de ce répertoire, la chaine « .. » indique le répertoire parent. Ensuite, le nom de fichier « foo » est ajouté. Chaque élément du chemin est séparé par « / ».

7.3 Pour renommer un fichier, il est possible d'utiliser l'opération « rename », ou de copier le fichier et ensuite effacer le fichier précédent. Quelle est la différence entre les deux méthodes?

La méthode basée sur la copie requiert le double de l'espace disque du fichier à renommer de manière temporaire. Tout le contenu du fichier est lu et réécrit. Si le fichier est copié sur le même disque, alors l'utilisation de « rename » change uniquement la chaine du nom du et/ou son répertoire parent, ce qui ne nécessite pas d'espace supplémentaire ni de recopie. Comme le contenu reste en place, c'est beaucoup plus efficace et rapide. Si le fichier de destination ne réside pas sur le même périphérique, rename retourne l'erreur EXDEV et il faut procéder avec la copie complète.

7.4 Un même fichier est ouvert depuis deux processus. L'un d'eux ferme le fichier et le supprime. Que se passera-t-il dans l'autre processus? À quel moment l'espace occupé sera-t-il libéré?

Le fichier existe tant qu'il est référencé par au moins un processus. Ainsi, même si le fichier n'a plus de nom dans aucun répertoire après la suppression, son contenu n'est pas libéré et il continue d'être disponible pour les processus qui l'utilisent encore. Lorsque le fichier n'est plus ouvert par aucun processus, alors l'espace est libéré. Il faut faire attention à cet aspect, car il peut manquer d'espace sur un disque à cause de gros fichiers temporaires qui restent ouverts et qui ne sont pas visibles dans l'arborescence.

7.5 Sous Windows, l'outil système « Defrag » compacte les blocs de fichiers sur un système de fichier FAT ou NTFS. Pourquoi est-ce utile sur un disque dur? Est-ce que cela diminue la fragmentation interne ou externe?

Le compactage améliore la contiguïté des fichiers et diminue le nombre de déplacements de la tête de lecture d'un disque dur, ce qui diminue la latence d'accès et augmente le débit de lecture. Les systèmes de fichiers FAT et NTFS allouent l'espace par blocs de taille fixe et donc ne sont pas sujets à une perte d'espace en raison de la fragmentation externe. Quant à la fragmentation interne, même si le fichier est formé de plusieurs blocs, seul le dernier est incomplet et cette situation ne change pas en réorganisant les blocs de manière contiguë.

7.6 Considérez une structure d'inode avec 4 pointeurs directs, un pointeur indirect de niveau 1 et un pointeur indirect de niveau 2. La taille des blocs est de 512 octets et celle des pointeurs est de 64 bits. Quelle est la proportion de l'espace qui est utilisé pour les données par rapport à l'ensemble des blocs destinés au fichier dans le cas d'un fichier de 64 Kio? Négligez l'espace occupé par l'inode.

Le nombre de blocs de données nécessaires pour stocker le fichier est 64 Kio / 0,5 Kio = 128 blocs. Les pointeurs directs stockent 4 blocs, ce qui reste 124 blocs à placer dans les autres niveaux. Le nombre de blocs adressables par le pointeur de niveau 1 est de 512 / 8 = 64, il reste donc 124 – 64 = 60 blocs à placer au niveau 2. Pour adresser 60 blocs depuis le pointeur indirect de niveau 2, il faut un bloc ayant une entrée vers un bloc qui contient 60 pointeurs vers les blocs de données. Le premier niveau indirect nécessite un bloc et le second niveau indirect nécessite 2 bloc, pour un total de 3. La proportion utilisée pour les données est donc 128 / (128 + 3) = 97,8%.

7.7 Considérez la structure d'inode de la question précédente. Calculez le temps moyen d'accès à chacun des blocs de données pour un fichier de taille maximale et des accès aléatoires dans le cas où il n'existe aucune cache des blocs, et que la lecture d'un bloc nécessite un délai d = 6 ms.

Les pointeurs directs référencent 4 blocs. Le premier niveau indirect référence 64 blocs. Le second niveau indirect référence 64 * 64 = 4096 blocs. Ainsi, la taille maximale est de 4164 blocs. Le temps d'accès pour les blocs directs nécessite la lecture de l'inode puis le bloc de données (2*d), le premier niveau indirect consiste en l'inode, le bloc indirect puis le bloc de données (3*d), et le second niveau indirect consiste à un accès de plus que le précédent (4*d). Pour la taille maximale et un accès aléatoire, le temps moyen d'accès est donc : (2*d*4/4164) + (3*d*64/4164) + (4*d*4096/4164), ce qui donne environ 23,9 ms.

7.8 Décrivez les avantages et les inconvénients d'un lien symbolique par rapport à un lien dur. Dans quels cas chacun d'eux est-il utilisé?

Un lien symbolique permet de créer des liens entre des fichiers sur deux systèmes de fichiers ayant leur propre liste d'inode tandis qu'un lien dur ne peut franchir la partition du fichier lié. Le lien dur a l'avantage de ne jamais être brisé,

contrairement à un lien symbolique qui le deviendra si le fichier lié est supprimé. Les liens symboliques sont généralement utilisés pour les liens entre des noms de librairies sur le système (voir /usr/lib), ce qui permet de déplacer une librairie sur une autre partition. Les liens durs peuvent être utilisés pour les sauvegardes, comme dans le système TimeMachine d'Apple.